Table of Contents

ActiveReports 14 User Guide

ActiveReports provides fully integrated Visual Studio components which combine user-friendly visual controls with the low-level control of code in Visual Studio .NET programming languages to provide a powerful report designer.

This is the help file for ActiveReports, reporting software for use in Visual Studio 2013, 2015, 2017, or 2019.

Topic	Content
Welcome to ActiveReports 14	Learn basic information on installing and using the product, as well as support, licensing, and what's new.
License Your ActiveReports	About licensing ActiveReports.
Quick Start	Overview to get started for designing reports.
Upgrade Reports	Upgrading reports from previous versions of ActiveReports and Data Dynamics Reports.
Import Reports	Importing MS Access Reports, Crystal Reports, Section Reports, and Excel files with the Import Wizard.
Report Types	Learn which type of ActiveReport best suits your needs.
Preview Reports	Previewing a report at design time or viewing it in Windows Form or Web Viewers.
Print Reports	Learn about printing.
Export Reports	Learn about rendering extensions and exports, and which formats are available with Page and Section reports.
Concepts	Basic concepts that you may need to create reports from scratch.
How To	Step-by-step instructions for common features.
Samples and Walkthroughs	Description of the samples available with ActiveReports and step-by-step walkthroughs explaining key features.
Troubleshooting	Provides troubleshooting symptoms, causes, and solutions to commonly encountered issues.
Class Library (on-line documentation)	API documentation with topics for all of the public members of each assembly included with ActiveReports.

Welcome to ActiveReports 14

Learn to use and install ActiveReports 14.

Topic	Content
What's New	Learn about the new features in ActiveReports.
ActiveReports Editions	Find out which features can be used with Standard and Professional Edition licenses.
Product Requirements	Learn about the hardware and software required to run ActiveReports.

Install ActiveReports	Find out how to install ActiveReports, the files provided by the setup, and Visual Studio Integration.
Available Packages	Learn about the packages available for ActiveReports through NuGet and NPM.
Manage ActiveReports Dependencies	Learn about managing packages available for ActiveReports through NuGet.
GrapeCity Copyright Notice	Explains GrapeCity copyright information.
End User License Agreement	Understand the terms of the ActiveReports License Agreement and Limited Warranty.
Redistributable Files	Find out the list of files that may be distributed.
Open Source Software	Lists Open Source software supported by ActiveReports.

What's New

We have made a number of changes and added new features since the last version of ActiveReports. Following are some of the major highlights of this release.

Enhancement in PDF Export (Pro)

With the new API, you can add metadata that includes document information properties like title, contributor, description, etc. to exported PDFs. You can also add files as attachments to exported PDFs. This feature is supported in the Pdf Rendering Extension and Pdf Export with Version PDF/A-3b or higher.

|Rendering Extension| |Export Filter|

Improved Excel Export

The Container, Table, Tablix and Subreport controls in Page and RDL reports have a new PageName property. This new property sets the name for a sheet with this report item in the exported Excel file.

.NET Core Support

The ActiveReports packages available through NuGet let you design and preview reports in .NET Core applications. The applications can be deployed in major operating systems such as Windows, Linux, and macOS.

Learn More

Improved Installation

ActiveReports 14 brings several improvements in the installation process to provide a simplified installation and update experience. The changes include:

- **ActiveReports Packages** are available for installation from NuGet, an open source package manager. All the dependencies will be automatically included.
- Latest **JavaScript** files and libraries for JSViewer and WebDesigner are available for download from NPM.
- Latest **samples** are available in the GitHub repository.

|Install ActiveReports| |Available Packages| |Samples|

New JSViewer (Pro)

An improved JSViewer in ActiveReports 14 offers more features in addition to the old JSViewer, which are Search panel, print without preview, and extensive export format support, including the ones for section reports.

Learn More

Improved Standalone Designer

The UI/UX enhancements have been made in the standalone designer shipped with ActiveReports 14 to comply with the modern look and feel.

Learn More

Improved WebViewer (Pro)

An improved WebViewer in ActiveReports 14 uses the Pro JSViewer instead of the old ASP.NET WebViewer, offering all advantages of using this new generation viewer features.

Learn More

InputField (Page and RDL Reports) (Pro)

A new report control for Page and RDL reports, **InputField**, provides support for editable fields in an exported PDF report where the InputField's value can be modified. The new control can be of two types, **Text** and **Checkbox**, that you can set in the **Type** property. Each selected type has its own set of properties.

Learn More

Rendering Extensions Improvements

In ActiveReports 14, we have improved the rendering extensions implementation to get better performance and user-friendly behavior. The API changes mainly affect the PDF and Image rendering extensions.

Learn More

Adding Data in Web Designer (Pro)

You can now access data directly from the ActiveReports 14 Web Designer by adding a data source and a data set under the Data tab. You can choose from a number of predefined data providers that include Microsoft SQL Client, Microsoft Ole DB, Microsoft ODBC, JSON, CSV, and XML.

Learn More

ActiveReports Editions

Available in two editions, Standard and Professional, ActiveReports 14 delivers outstanding reporting capabilities. Drop

down the sections below to see the features packed into each edition.

Standard Edition Features

The Standard Edition provides a report designer that is fully integrated with the Visual Studio IDE, a report viewer for Windows Forms, and export filters for generating reports in various file formats. The report designer even includes a barcode control with all of the most popular barcode styles, and its own chart control.

Designer

- Full integration with the .NET environment
- Familiar user interfaces
- Choice of section or page or RDL report types
  - C# and VB.NET support with code-based section reports
  - Script support with XML-based section reports
  - Expression support with page reports and RDL reports
- The ability to compile reports into the application for speed and security or to keep them separate for ease of updating
- Designer hosting of .NET and user controls

Report Controls

Section Reports	Page Reports/RDL Reports
• ReportInfo • Label • Line • PageBreak • OleObject • SubReport • Shape • Picture • RichTextBox with HTML tag support • Chart with separate data source • Textbox • Barcode with standard styles plus RSS and UPC styles • Checkbox • CrossSectionBox extends from a header section to the related footer section • CrossSectionLine extends from a header section to the related footer section	• Table data region • Tablix data region • Map data region • Chart data region • List data region • BandedList data region • Sparkline data region • FormattedText with mail merge capabilities and XHTML + CSS support • Bullet Graph • BarCode • CheckBox • TextBox • TableOfContents • Line • Container • Shape • Image • Subreport • Overflow Placeholder

Expressions (Page reports/RDL reports)

- Aggregates
- Data visualization
  - Data bar
  - Icon set
  - Range bar
  - Color scale
  - Gradient
  - Hatch

Interactive Features

- Document map (table of contents)
- Bookmark links, hyperlinks, and drill through links
- Parameters
- Drill-down (page report/RDL reports)
- Copy, pan, and zoom
- Jump to previous, next, first, or last group or search result

Reporting Engine

- Managed code
- Binding to ADO.NET, XML, iList, and custom data sources
- Master reports, themes, and styles
- All of the features of previous versions of ActiveReports and Data Dynamics Reports

Windows Forms Report Viewer

- Managed C# code
- Very small deployment assembly, suitable for use on the Internet
- Table of contents and bookmarks
- Thumbnail view
- HyperLinking
- Annotations (section reports only)
- Configurable scroll bar jump buttons (like those found in Microsoft® Word®)
- Parameters
- Bookmark links, hyperlinks and drillthrough links
- Interactive sorting (page reports/RDL reports)
- Touch mode

WPF Viewer

- Managed C# code
- Table of contents and bookmarks
- Thumbnail view
- Parameters
- Annotations
- Configurable scroll bar jump buttons (like those found in Microsoft® Word®)
- Bookmark links, hyperlinks and drillthrough links

- Interactive sorting

Export Filters

ActiveReports includes export filters to generate output into many popular formats.

Export formats	Section report	Page/RDL report
Html: Export reports to HTML, DHTML, or MHT formats, all of which open in a Web browser.	✓	✓
Pdf: Export reports to PDF, a portable document format that opens in the Adobe Reader.	✓	✓
Rtf: Export reports to RTF, RichText format that opens in Microsoft Word, and is native to WordPad.	✓	✓
Word: Export reports to DOC, a format that opens in Microsoft Word.	✘	✓
Text: Export reports to TXT, plain text, a format that opens in Notepad or any text editor. Export reports to CSV, comma separated values, a format that you can open in Microsoft Excel.	✓	✓
Image: Export reports to BMP, GIF, JPEG, or PNG image format.	✘	✓
Tiff: Export reports to TIFF image format for optical archiving and faxing.	✓	✓
Excel: Export reports to formats that open in Microsoft Excel, XLS or XLSX.	✓	✓
Xml: Export reports to XML, a format that opens in a Web browser or delivers data to other applications.	✘	✓
CSV: Export reports to a CSV file, a form of structured data in plain text. The text in a CSV file is saved as series of values separated by comma.	✘	✓
JSON: Export reports to a JSON file, a text-based data format in which the data is stored in the hierarchical form.	✘	✓

Import Filters

- Access® Reports
- Crystal Reports
- Excel file
- RPX file

Stand-Alone Applications

- The Report Designer application lets you create report layout, define data, and add interactive features report, and lets you preview the reports and export or print them. It can be opened from the shortcut provided in the Start menu.
- The Report Viewer application contains all the functionality of the ReportPreview control. It can be opened from the shortcut provided in the Start menu.
- The WPF Viewer application contains all the functionality of the WPF Viewer control.
- The ActiveReports Import Wizard application allows importing Microsoft Access reports, Crystal Reports, Excel files, and RPX files. It can be opened from the shortcut provided in the Start menu.

Professional Edition Features

The Professional Edition includes all of the features of the Standard Edition and supports the following additional features:

End-User Report Designer

The control is a run-time designer that may be distributed royalty-free. It allows the ActiveReports designer to be hosted in an application and provides end-user report editing capabilities. The control's methods and properties provide easy access for saving and loading report layouts, monitoring and controlling the design environment, and customizing the look and feel to the needs of end users.

ASP.NET Integration

- The Web server control provides convenience for running and exporting reports in ASP.NET.
- HTTP Handler extensions allow report files (RPX or RDLX) or compiled assemblies containing reports to be dropped on the server and hyperlinked.

WebViewer Control

- The WebViewer control allows quick viewing of ActiveReports on the web as well as printing capability with the AcrobatReader ViewerType enumeration.

HTTP Handlers

- The RPX and RDLX HTTPHandler allows the developer to hyperlink ActiveReports on a web page to return HTML format or PDF format reports for viewing and/or printing.
- The Compiled Report HTTPHandler allows the developer to hyperlink ActiveReports compiled in an assembly on a web page to return HTML format or PDF format reports for viewing and/or printing.

Table of Contents Control

- TableOfContents control is used to display the document map, an organized hierarchy of the report heading levels and labels along with their page numbers, in the body of a report.
- TableOfContents control allows you to quickly understand and navigate the data inside a report in all viewers that are supported in ActiveReports

Map Control

- The Map data region shows your business data against a geographical background.
- Create different types of map, depending on the type of information you want to communicate in your report.

PdfSignature and TimeStamp Features

- The PdfSignature class allows you to provide PDF document digital signatures and certification.
- The PdfStamp class allows you to draw the digital signatures and certification onto the documents.
- The TimeStamp class allows you to add a TSA (Time Stamping Authority) stamp to your digital signatures.

Font Linking

- Font linking helps you resolve the situation when fonts on a deployment machine do not have the glyphs that were used in a development environment.
- By linking fonts, you can resolve the problem with a different PDF output on deployment and development machines that may occur due to the missing glyphs.

Font Fallback

- If missing glyphs are not found in linked fonts, the PDF export filter looks for the glyphs in fonts declared in the FontFallback property.
- A default font is used if you do not declare one, or you can declare an empty string for this property to leave out missing glyphs from the exported file.

PDF Export

- PDF/A and PDF/UA support.
- IVS character support.
- Devanagari character support.

Bold Font Emulation (PDF Export Filter)

- Some fonts (for example, Franklin Gothic Medium, Microsoft Sans Serif, most East Asian fonts, etc.) may lose bold style for the PDF output. The Professional Edition provides bold style emulation in the PDF export filter to eliminate this limitation.

Word Export

- Export your reports in .docx format, a format that opens in Microsoft Word application.

Web Designer

- Create or modify reports by embedding Web Designer into your web applications.

JSViewer

- View reports in all modern browsers using JSViewer.

InputField Control

- The InputField report control provides support for editable fields in an exported PDF report where the InputField's value can be modified.
- Choose one of the two report contorl types – Text and Checkbox. Each selected type has its own set of properties.

Comparison Between Editions

Professional Edition features are disabled or marked with an evaluation banner if you have purchased a Standard Edition license.

Features		Standard	Professional
Visual Studio Controls			
Web Forms	**WebViewer**: Use this control to display your reports on the Web. Includes viewer types HTML and PDF.	✘	✓
	HTTP Handlers: PDF and HTML (compiled report, RPX file)	✘	✓
Windows Forms	**Viewer**: Use this control to offer your users report zoom and preview, multiple tabs for hyperlinks, split-page and multi-page views, a Table of Contents pane, a Thumbnails pane, text searches, and annotations.	✓	✓
	Designer: Use this control to create a royalty-free, custom designer that your end users can use to create and modify their own reports.	✘	✓
	ReportExplorer: Use this control along with the Designer control to provide functionality to your users.	✓	✓
	ToolBox: Use this control along with the Designer control to provide report controls for your users.	✓	✓
	LayerList: Use this control along with the Designer control to provide Layers functionality to your users.	✓	✓
	ReportsLibrary: Use this control to view, add, or hide report parts.	✓	✓
WPF	**WPF Viewer**: Use this control to display your section, page and RDL reports. The WPF Viewer offers the Thumbnails pane, the Parameters pane, the Document map pane, and the Search results pane.	✓	✓
Web and Windows Forms	**HtmlExport**: Export reports to HTML, DHTML, or MHT formats that open in a Web browser.	✓	✓
	PdfExport: Export reports to PDF, a portable document format that opens in the Adobe Reader.	✓	✓
	RtfExport: Export reports to RTF, RichText format that opens in Microsoft Word, and is native to WordPad.	✓	✓
	WordExport (.doc): Export reports to DOC (Word HTML), a format that opens in Microsoft Word.	✓	✓
	WordExport (.docx): Export reports to DOCX (LibreOffice), a format that opens in any word processing software.	✘	✓
	TextExport: Export reports to TXT, plain text, a format that opens in Notepad or any text editor. This export filter can also export reports to CSV, comma separated values, a format that you can open in Microsoft Excel.	✓	✓
	ImageExport: Export reports to BMP, GIF, JPEG, TIFF, or PNG image format. Note that you can only export section reports to the TIFF image type. All other	✓	✓

	image types are for page reports and RDL reports.		
	XlsExport: Export reports to formats that open in Microsoft Excel, XLS or XLSX.	✓	✓
	XmlExport: Export reports to XML, a format that opens in a Web browser or delivers data to other applications.	✓	✓
Components	**Web Designer**: Create or modify reports by embedding Web Designer into your web applications.	✘	✓
	JSViewer: View reports in all modern browsers using JSViewer.	✘	✓
PDF Export Advanced Features	Digital signatures	✘	✓
	Time stamp	✘	✓
	EUDC	✘	✓
	Select from Japanese embedded fonts or unembedded fonts	✘ *1	✓
	Bold	✘	✓
	Italic	✓	✓
	Multi Language	✓ *2	✓
	PDF/A and PDF/UA Support	✘	✓
	IVS Character Support	✘	✓
	Devanagari Character Support	✘	✓
	Print Presets	✘	✓
Integrated Report Designer			
Design Format	Section reports support banded layouts. Page reports support fixed page layouts. RDL reports support continuous page layout.	✓	✓
Script and Code	In section reports, you can add C# or VB code to events behind your code-based reports, or add script to events in the script editor in XML-based reports. In page reports/RDL reports, you can use regular expressions in any property, plus you can add VB.NET methods to the code tab, and call them in your expressions.	✓*3	✓*3
Report File Formats	You can save and load page reports/RDL reports in RDLX (extended RDL) format. You can save and load section reports in RPX (report XML) format, and you can compile section reports in CS or VB code formats.	✓	✓
Report Controls	The **BarCode** control supports all of the following styles: ANSI 3 of 9 | ANSI Extended 3 of 9 | Code 2 of 5 | Interleaved 2 of 5 Code 25 Matrix | Code 39 | Extended Code 39 | Code 128 A Code 128 B | Code 128 C | Code 128 | Code 93	✓	✓

		Auto			
Extended Code 93	MSI	PostNet	Codabar		
EAN-8	EAN-13	UPC-A	UPC-E0		
UPC-E1	RoMail RM4SCC	UCC/EAN-128	QRCode		
Code 49	Japanese Postal	Pdf417	EAN-128 FNC1		
RSS-14	RSS-14 Truncated	RSS-14 Stacked	MicroPdf417		
RSS-14 Stacked Omnidirectional	RSS Expanded	RSS Expanded Stacked	MicroQRCode		
BC412	Code_11	ISBN	ISMN		
ISSN	ITF14	MaxiCode	Pharmacode		
Plessey	PZN	SSCC_18	Telepen		
IntelligentMail	IntelligentMailPackage	HIBC Code 128	HIBC Code 39		

The **InputField** control provides support for editable fields in an exported PDF report where the InputField's value can be modified.	✘	✓
The **Map** control allows you to display data against a geographical background on the report.	✘	✓
The **TableofContents** control allows you to display a document map in an organized hierarchy of the report heading levels and labels along with there page numbers, in the body of a report.	✘	✓

The **Chart** control supports all of the following styles:
• **Common Charts**: Area, Bar2D, Bezier, Doughnut/Pie, Line, Scatter, StackedArea, StackedBar, StackedArea100Pct, and StackedBAR110Pct
• **3D Charts**: Area3D, Bar3D, ClusteredBar, Line3D, Doughnut3D/Pie, StackedBar3D, and StackedBar3D100Pct
• **XY Charts**: Bubble, BubbleXY, LineXY, and PlotXY
• **Financial Charts**: Candle, HiLo, and HiLoOpenClose
• Composite Charts for following chart types:
  ◦ Column: Plain, Stacked, Percent Stacked
  ◦ Area: Plain, Stacked, Percent Stacked
  ◦ Line: Plain, Smooth
	✓	✓

Other report controls include: Label, TextBox, CheckBox, Picture Line, Shape, RichText, PageBreak	✓	✓

	SubReport	ReportInfo	CrossSectionLine	CrossSectionBox		
Styles and Report Settings	You can control page settings, printer settings, global settings such as grid display, grid size, and whether to show a verification dialog when deleting controls. You can specify row count or column count in grids, ruler units, and how many pages to display in previews.				✓	✓
External Style Sheets	You can reuse report designer styles by saving and loading style information in external files.				✓	✓
Others	The designer also offers snaplines, report preview, designer zoom, various formatting settings, control and text alignment settings, Z order settings, unbound fields, and parameters support.				✓	✓
Input and Output						
Data	Supported data includes: ADO.NET data provider, ADO.NET data class (DataSet, DataTable, DataReader, DataView), XML data, and unbound data				✓	✓
Print	You can control the page size, orientation, and margins, as well as specifying bound (double page spread), collating, duplex printing, and paper feed trays.				✓	✓
Import	You can import Crystal Reports, MS Access Reports, Excel files, and RPX files using the ActiveReports Import Wizard.				✓	✓

*1: Japanese fonts can only be output as embedded fonts.
*2: Cannot handle output of multiple language fonts in a single control. Please refer to Multi-Language PDF for details.
*3: See Designing Code-based Section Reports in .NET Core for more information.

Product Requirements

To install and use ActiveReports 14, you need the following hardware and software.

Hardware requirements (minimum)

- **Hard drive space**: 200 MB available

Development Environments

	.NET Framework	.NET Core
Application	Windows Forms, WPF, ASP.NET Web Forms, ASP.NET MVC5, JavaScript libraries (JSViewer, Web Designer)	Windows Forms, WPF, ASP.NET Core MVC, JavaScript libraries (JSViewer, Web Designer)
Operating System	Windows 8.1, or 10, or Windows Server 2012, 2012 R2, 2016, and 2019	Windows 8.1, or 10, or Windows Server 2012, 2012 R2, 2016, and 2019
Microsoft Visual Studio	2013, 2015, 2017, and 2019	2019 (16.4+)
Internet Information	8.0, 8.5, 10	8.0, 8.5, 10

Services		

> **Note:**
> The Express Editions of Visual Studio do not work with ActiveReports, as they do not support packages.

You are required to perform following updates from the links provided for proper working in Visual Studio.

Developer Pack for .NET Framework 4.6.2 or above

https://www.microsoft.com/download/visual-studio-sdks

NuGet Package

- for VS2013
  https://marketplace.visualstudio.com/items?itemName=NuGetTeam.NuGetPackageManagerforVisualStudio2013
- for VS2015
  https://dist.nuget.org/visualstudio-2015-vsix/latest/NuGet.Tools.vsix

Visual Studio Update Packages

- update 5 for VS2013
  https://docs.microsoft.com/en-us/visualstudio/releasenotes/vs2013-update5-vs
- Update 3 for VS2015
  https://docs.microsoft.com/en-us/visualstudio/releasenotes/vs2015-update3-vs
- update 3 and KB3165756 for VS2015
  https://docs.microsoft.com/en-us/previous-versions/mt752379(v=vs.140)
- version 15.9 for VS2017
  https://docs.microsoft.com/en-us/visualstudio/releasenotes/vs2017-relnotes
- version 16.5 for VS2019
  https://docs.microsoft.com/en-us/visualstudio/releases/2019/release-notes#16.5.0

TypeScript Plugin

Download latest TypeScript plugin (required for Angular applications)

- For VS2015
  https://www.microsoft.com/en-us/download/details.aspx?id=48593
- For VS2017/VS2019
  https://github.com/Microsoft/TypeScript/releases

Run Time Supported Environments

Windows Forms and WPF

- Microsoft® .NET Framework Version: 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8 or Microsoft® .NET Core 3.1
- Operating System:
  Windows 8.1, or 10, or
  Windows Server 2012, 2012 R2, 2016, and 2019

Web Development

	.NET Framework	.NET Core
Server		

Version	Microsoft® .NET Framework Version 4.6.2 and above	Microsoft® .NET Core 3.1
Application	ASP.NET Web Forms, ASP.NET MVC5, JavaScript libraries (JSViewer, Web Designer)	ASP.NET Core MVC, JavaScript libraries (JSViewer, Web Designer)
Operating System	Windows 8.1, or 10, or Windows Server 2012, 2012 R2, 2016, and 2019	Windows 8.1, or 10, or Windows Server 2012, 2012 R2, 2016, and 2019 macOS 10.13+, RHEL 6+, Fedora 29+, Ubuntu 16.04+, Debian 9+, SLES 12 SP2+
Internet Information Services	8.0, 8.5, 10	-
Client		
Browser	Microsoft Internet Explorer 11 Microsoft Edge Google Chrome (v50 or higher) Mobile Safari Firefox (v57 and above) Safari (macOS 10 and higher) Safari (iOS 10 and higher) Safari (iPadOS 13)	

Limitations of .NET Core support:

- Since there is no design-time support for Code-Based Section reports in WinForms .NET Core projects, you need to link reports and use .NET Framework WinForms Designer for the design-time experience in .NET Core. See Designing Code-based Section Reports in .NET Core for more information.
- You need to register encodings before using ActiveReports with ASP.NET Core MVC. See how to register encodings in Troubleshooting page.
- You need to update few references for some Microsoft Data Providers that do not appear in DataSource Editor in .NET Core. See how to resolve this in Troubleshooting page.
- While working with Section reports with scripts in WinForms Viewer, WPF Viewer, and Windows Designer components in .NET Core applications, it is mandatory to install 'System.Text.Encoding.CodePages' nuget package and update the Program.cs file accordingly. Otherwise 'System.NotSupportedException' is thrown on previewing the report. For more information, see Troubleshooting page.

Install ActiveReports

This topic elaborates the steps to install ActiveReports 14, lists the files that are installed, and about Visual Studio Integration.

Installing ActiveReports 14 using MSI file

The complete setup of ActiveReports 14 can be installed using MSI file.

Before you begin with the following steps, download the installer (ActiveReports-v14.x.x.0.msi) from the website and follow these steps.

1. Double-click the ActiveReports-v14.x.x.0.msi file or right-click the file and select **Install**.
2. On the End-User License Agreement screen that appears, go through the terms in the License Agreement.
    - If you want to continue with the default installation settings, select the check box to accept the license agreement and click **Install** to continue with installation.
    - If you want to change the default installation path and the way product features are installed, do the following:

        1. Select **Advanced** and click **Change**.
        2. Specify the path for the installation and click OK.
        3. Click **Next** and select how controls, features, and samples are installed and integrated with Visual Sudio versions.
        4. Select **Install**.
3. On the User Account Control screen, select Yes to allow ActiveReports software installation on your PC.
4. Once the installation finishes, a screen notifying the completion of installation appears. Select **Activate now** check box and click **Finish** to close the window and complete the installation process.
5. Click **Yes** on the User Account Control screen to continue with activating the license. See License Your ActiveReports for information on licensing.

Installed Files

You can verify the installation via installer by following the steps below:

1. Open Visual Studio.
2. From the Visual Studio **Help** menu, select **About Microsoft Visual Studio** and verify that **ActiveReports 14** appears in the installed products list.

When you run the installer (ActiveReports-v14.x.x.0.msi), you get necessary utilities such as standalone designer and viewer applications, import tool, and license manager, along with NuGet packages, localization resources, etc. If you use all of the default settings, files are installed in the following folders:

C:\ProgramData\Microsoft\Windows\Start Menu\Programs\GrapeCity

File (or Folder)	Description
ActiveReports (folder)	Shortcut to the folder containing stand-alone applications and Samples folder. See the next dropdown for further details.
GrapeCity License Manager	Shortcut to the License Manager application.

C:\ProgramData\Microsoft\Windows\Start Menu\Programs\GrapeCity\ActiveReports 14

File (or Folder)	Description
ActiveReports 14 Designer	Shortcut to the stand-alone designer application.
ActiveReports 14 Import	Shortcut to the ActiveReports Import wizard application.
ActiveReports 14 Theme Editor	Shortcut to the ActiveReports Theme Editor application.

ActiveReports 14 User Guide(CHM)	Shortcut to the Compiled HTML help file for ActiveReports.
ActiveReports 14 User Guide(PDF)	Shortcut to the PDF help file for ActiveReports.
ActiveReports 14 User Guide(Web)	Shortcut to the Online help file for ActiveReports.
ActiveReports 14 Viewer	Shortcut to the stand-alone ActiveReports Viewer application.

C:\Program Files\GrapeCity\ActiveReports 14 (C:\Program Files (x86)\GrapeCity\ActiveReports 14 on a 64-bit Windows operating system)

Folder	Description
Deployment	Includes templates for WPF.
Icons	Includes associated Icons image files.
Localization	Includes Resource and DOS batch files for localizing ActiveReports components.
NuGet	Contains NuGet packages. See Available Packages for more information.
Tools	Includes all ActiveReports application, XML configuration files, and localization resources for designer and import tool in Chinese environment.
VisualStudio	Includes Microsoft Visual Studio application extension and bootstrapper application.
GrapeCity.ActiveReports.config	XML configuration file.

C:\ProgramData\GrapeCity\gclm

File or Application	Description
gclm.exe	License Manager setup file.
gclm.exe.config	License Manager setup XML configuration file.

C:\Program Files\Grapecity\ActiveReports 14\Tools (C:\Program Files (x86)\GrapeCity\ActiveReports 14\Tools on a 64-bit Windows operating system)

File or Application	Description
zh-CN folder	Includes resource files for the Chinese environment.
GrapeCity.ActiveReports.Designer.exe	Stand-alone designer setup file.
GrapeCity.ActiveReports.Designer.exe.config	Stand-alone designer setup XML configuration file.
GrapeCity.ActiveReports.Designer.VisualElementsManifest.xml	Stand-alone designer's .VisualElementsManifest xml file.
GrapeCity.ActiveReports.Imports.exe	ActiveReports Import application setup file.
GrapeCity.ActiveReports.Imports.exe.config	ActiveReports Import application setup XML configuration file.
GrapeCity.ActiveReports.Imports.Win.exe	ActiveReports Import wizard setup file.
GrapeCity.ActiveReports.Imports.Win.exe.config	ActiveReports Import wizard setup XML configuration file.

GrapeCity.ActiveReports.ThemeEditor.exe	ActiveReports Theme Editor setup file.
GrapeCity.ActiveReports.ThemeEditor.exe.config	ActiveReports Theme Editor setup XML configuration file.
GrapeCity.ActiveReports.Viewer.exe	Stand-Alone ActiveReports Viewer setup file.
GrapeCity.ActiveReports.Viewer.exe.config	Stand-Alone ActiveReports Viewer setup XML configuration file.
GrapeCity.ActiveReports.WpfViewer.exe	Stand-Alone ActiveReports WPF Viewer setup file.
GrapeCity.ActiveReports.WpfViewer.exe.config	Stand-Alone ActiveReports WPF Viewer setup XML configuration file.
GrapeCity.ActiveReports.Chart.dll	Chart control assembly file.
GrapeCity.ActiveReports.Core.DataProviders.dll	Data Providers assembly.
GrapeCity.ActiveReports.Core.Drawing.Gc.dll	Drawing Gc assembly file.
GrapeCity.ActiveReports.Core.Drawing.Gdi.dll	Drawing Gdi assembly file.
GrapeCity.ActiveReports.Export.Excel.dll	Excel Export assembly file.
GrapeCity.ActiveReports.Core.Export.Excel.Page.dll	Excel Export (for Page report) assembly file.
GrapeCity.ActiveReports.Export.Html.dll	HTML Export assembly file.
GrapeCity.ActiveReports.Core.Export.Html.Page.dll	HTML Export (for Page report) assembly file.
GrapeCity.ActiveReports.Export.Image.dll	Image Export assembly file.
GrapeCity.ActiveReports.Core.Export.Image.Page.dll	Image Export (for Page report) assembly file.
GrapeCity.ActiveReports.Export.Pdf.dll	HTML Export assembly file.
GrapeCity.ActiveReports.Core.Export.Pdf.Page.dll	HTML Export (for Page report) assembly file.
GrapeCity.ActiveReports.Core.Export.Svg.Page.dll	Svg Export assembly file.
GrapeCity.ActiveReports.Core.Export.Text.Page.dll	Text Export (for Page report) assembly file.
GrapeCity.ActiveReports.Export.Word.dll	Word Export assembly file.
GrapeCity.ActiveReports.Core.Export.Word.Page	Word Export (for Page report) assembly file.
DocumentFormat.OpenXml.dll	OpenXML assembly file.
Gcef.Data.DataEngine.dll	Gcef's DataEngine assembly file.
Gcef.Data.ExpressionInfo.dll	Gcef's ExpressionInfo assembly file.
Gcef.Data.VBFunctionLib.dll	Gcef's VBFunctionLib assembly file.
GrapeCity.ActiveReports.Design.Win.dll	Windows Designer assembly file.
GrapeCity.ActiveReports.Document.dll	Document assembly file.
GrapeCity.ActiveReports.Export.Rdf.dll	RDF Export assembly file.
GrapeCity.ActiveReports.Export.Xml.dll	XML Export assembly file.
GrapeCity.ActiveReports.Interop.dll	Native functions assembly file.

GrapeCity.ActiveReports.Core.Rdl.dll	RDL assembly file.
GrapeCity.ActiveReports.Core.Rendering.dll	Rendering assembly file.
GrapeCity.ActiveReports.dll	Run time engine assembly file.
GrapeCity.ActiveReports.Viewer.Win.dll	Windows Viewer assembly file.
GrapeCity.ActiveReports.Viewer.Wpf.dll	WPF Viewer assembly file.
GrapeCity.DataVisualization.dll	Data visualization assembly file.
GrapeCity.Documents.Common.dll	Assembly file used by Documents PDF.
GrapeCity.Documents.Common.Windows.dll	Assembly file required for providing support for font linking specified in the Windows registry and access to native Windows imaging APIs, improving performance and adding some features (e.g. TIFF support)
GrapeCity.Documents.DX.Windows.dll	Assembly file used by GrapeCity.Documents.Common.Windows.dll.
GrapeCity.Documents.Imaging.dll	Assembly file for image handling.
GrapeCity.Documents.Pdf.dll	Documents Pdf assembly file to handle the PDFs.
System.IO.Compression.dll	Compression/decompression assembly file.
GrapeCity.ActiveReports.Imports.Access.dll	Microsoft Access Import assembly file.
GrapeCity.ActiveReports.Imports.Crystal.dll	Crystal Reports Import assembly file.
GrapeCity.ActiveReports.Imports.Excel.dll	Excel Import assembly file.
GrapeCity.ActiveReports.Imports.RPX.dll	RPX Import assembly file.

Note: Samples and data used in the samples are not included with the installer. You need to download them separately from GitHub.

Visual Studio Integration

The ActiveReports installation provides you with following Visual Studio Integrations that help you configure projects and design reports.

Project Templates

Create a report layout using built-in sample application templates:

- ActiveReports 14 Page Report Application
- ActiveReports 14 RDL Report Application
- ActiveReports 14 Section Report Application (xml-based)
- ActiveReports 14 Section Report Application (code-based)
- ActiveReports 14 JSViewer Core MVC Application
- ActiveReports 14 JSViewer MVC Application

Item Templates

Add following item templates to your project:

- ActiveReports 14 Page Report
- ActiveReports 14 RDL Report
- ActiveReports 14 Section Report (xml-based)
- ActiveReports 14 Section Report (code-based)

Toolbox with Report Controls

A new ActiveReports 14 tab is automatically added in the toolbox with the controls in sync with the references.

Integrated Designer

ActiveReports offers an integrated designer that lets you create report layouts in Visual Studio and edit them at design time, visually, and through code and scripts. Along with the designer, you can use:

- **Report Explorer** to view the report elements in tree view,
- **Group Editor** to manage grouping in Tablix data region,
- **Layers List** to view layers in a report,
- **Reports Library** to view list of reports and controls added in the reports.

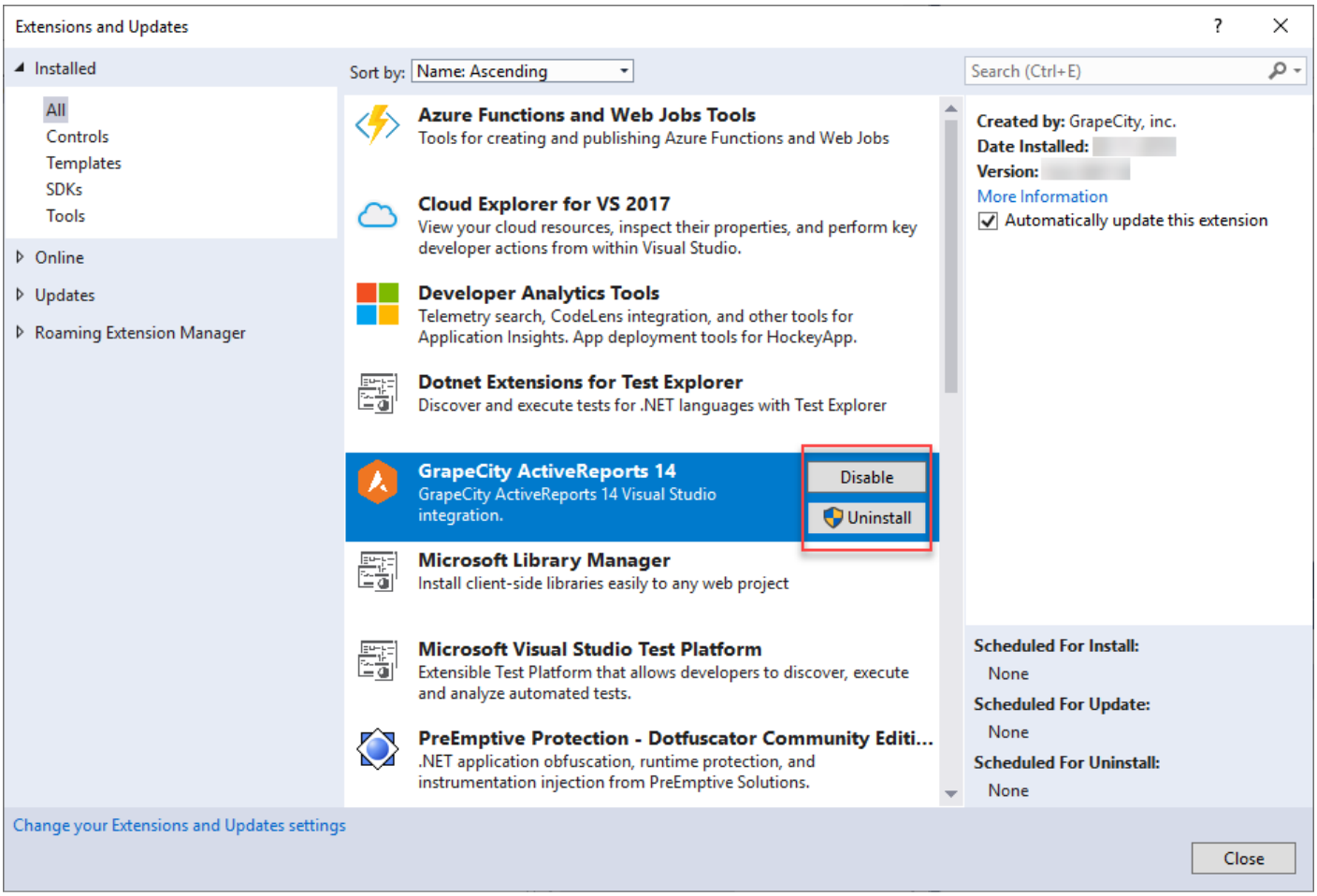
Conversion Tool

Use the conversion or upgrade tool - **Convert to ActiveReports 14** - to easily upgrade to the latest version. This tool is available under **Tools** menu option of Visual Studio.

To Remove Visual Studio Integration

In case you want to disable or uninstall GrapeCity ActiveReports 14 Visual Studio integration, do following.

1. Go to **Tools** menu and click **Extensions and Updates**
   OR
   In Visual Studio 2019, go to **Extensions** > **Manage Extensions**.
2. Navigate to **GrapeCity ActiveReports 14** and click Disable or Uninstall.



In case you want to install the integration again, follow these steps.

1. Go to the installation folder - C:\Program Files (x86)\GrapeCity\ActiveReports 14\VisualStudio.
2. Run the VSIX as per your Visual Studio Version.

Note:

- It is not possible to install ActiveReports 14 and ActiveReports 13 side-by-side.
- It is not possible to use different versions of ActiveReports for .NET within a single Visual Studio project.
- If you are creating an ActiveReports 14 application in Visual Studio 2017 or Visual Studio 2019, and you also have ActiveReports 13 installed, then you may get few errors in the property grid such as 'Object does not match target type' or 'Object reference not set to an instance of Object'. This issue is related to Visual Studio; you just need to restart Visual Studio to resolve the issue. Before restart, save the project, if necessary.
- When using our Integrated Designer in Visual Studio 2019, you may observe invisible controls in our smart panels or tool windows (such as Report Explorer or Layers List). To solve this issue, navigate to Tools > Options > Environment > General and clear the 'Optimize rendering for screens with different pixel densities' checkbox.

Available Packages

Following packages are available for download.

NPM

Download the packages for viewer and designer from **NPM**, a JavaScript package manager .

Reference	Purpose
ActiveReports JSViewer	Provides the report viewer based on JavaScript.
ActiveReports WebDesigner	Provides the report designer based on HTML5/JS technology stack.

NuGet

Download the following packages from **NuGet**, an open source package manager.

Reference	Purpose
GrapeCity.DataVisualization	Adds all the references that enable you to use Data visualization assembly file.
Grapecity.ActiveReports	GrapeCity ActiveReports is a set of assemblies that enable you to create, render, print, and export reports in a .NET or .NET Core application. This package includes the core engine assemblies required to create and render pixel perfect, WYSIWYG, reports.
Grapecity.ActiveReports.Export.Xml	This package includes assemblies needed to export ActiveReports files to XML.
Grapecity.Activereports.Export.Image	This package includes assemblies needed to export ActiveReports files to Image.
Grapecity.Activereports.Export.Word	This package includes assemblies needed to export ActiveReports files to Word (Doc/Docx).
Grapecity.Activereports.Export.Html	This package includes assemblies needed to export ActiveReports files to HTML.
Grapecity.Activereports.Chart	This package includes the core engine assemblies required to create and render pixel perfect, WYSIWYG, reports.
Grapecity.Activereports.Export.Rdf	GrapeCity ActiveReports RDLX->RDF bridge components.
Grapecity.Activereports.Export.Pdf	This package includes assemblies needed to export ActiveReports files to PDF.
GrapeCity.ActiveReports.Design.Win	This package includes assemblies needed to design reports and report layouts. Multiple designers are included for your convenience. Among them are Windows Forms (desktop) based designer, Visual Studio integrated designer, and JavaScript, and web based designer.
Grapecity.Activereports.Document	This package includes the core engine assemblies required to create and render pixel perfect, WYSIWYG, reports.
Grapecity.Activereports.Export.Excel	This package includes assemblies needed to export ActiveReports files

	to Excel.
Grapecity.Activereports.Core.Export.Image.Page	ActiveReports Image export library.
Grapecity.Activereports.Core.Export.Html.Page	ActiveReports HTML export library.
Grapecity.Activereports.Core.Export.Pdf.Page	ActiveReports Pdf export library.
Grapecity.Activereports.Core.Drawing.Gc	ActiveReports rendering with GcDocs implementation.
Grapecity.Activereports.Core.Export.Text.Page	ActiveReports JSON and CSV export library.
Grapecity.Activereports.Core.Rendering	ActiveReports common rendering implementation.
Grapecity.Activereports.Core.Export.Svg.Page	ActiveReports SVG export library.
Grapecity.Activereports.Core.Export.Excel.Page	ActiveReports Excel export library.
Grapecity.Activereports.Core.Export.Word.Page	ActiveReports Word export library.
Grapecity.Activereports.Core.Drawing.Gdi	ActiveReports rendering with GDI+ implementation.
Grapecity.Activereports.Core.Rdl	ActiveReports report template object model definition.
Grapecity.Activereports.Core.DataProviders	ActiveReports data providers facility.
Grapecity.ActiveReports.Aspnet.Viewer Grapecity.ActiveReports.Aspnetcore.Viewer	GrapeCity ActiveReports is a set of assemblies that enable you to create, render, print, and export reports in a .NET or .NET Core application. The packages include assemblies needed to display reports in ASP.NET report viewer. The viewer is customizable and includes printing and exporting features.
Grapecity.ActiveReports.Web	GrapeCity ActiveReports is a set of assemblies that enable you to create, render, print, and export reports in a .NET or .NET Core application. This package includes assemblies needed to display reports in report viewers. Multiple viewers are included for your convenience. Among them are viewers for Windows Forms, WPF, ASP.NET, and JavaScript. These viewers are customizable and include printing and exporting features.
Grapecity.ActiveReports.Web.Viewer	GrapeCity ActiveReports is a set of assemblies that enable you to create, render, print, and export reports in a .NET or .NET Core application. This package includes internal assemblies needed to display reports in report viewers.
Grapecity.ActiveReports.Aspnet.Designer Grapecity.ActiveReports.Aspnetcore.Designer	GrapeCity ActiveReports is a set of assemblies that enable you to create, render, print, and export reports in a .NET or .NET Core application. Th packages include assemblies needed to design reports and report layouts. Multiple designers are included for your convenience. Among them are Windows Forms (desktop) based designer, Visual Studio integrated designer, and JavaScript (web) based designer.
Gcef.Data.DataEngine	GrapeCity data processing engine.
Gcef.Data.ExpressionInfo	GrapeCity expression evaluation.

Gcef.Data.VBFunctionLib	GrapeCity VB functions implementation.

Manage ActiveReports Dependencies

The ActiveReports dependencies in Visual Studio projects can be added, updated, or removed using **NuGet** package manager. How the dependencies are managed depends on the location of package - at a local source or public source. The ActiveReports packages are available in the local directory on installing ActiveReports using the installer. The packages are also available publically on the **NuGet** website. For the list of packages, see Available Packages.

The following sections elaborate adding the ActiveReports assemblies in your project by either installing NuGet packages from local source or directly from the NuGet website.

Installing Packages from Local Source

You must first run the installer to obtain the **NuGet** packages locally - **C:\Program Files (x86)\GrapeCity\ActiveReports 14\NuGet**. See Install ActiveReports for the steps on installation using MSI file. Then, create the NuGet package source to add the NuGet feed URL into your NuGet settings in Visual Studio, as follows:

1. Configure local NuGet package source

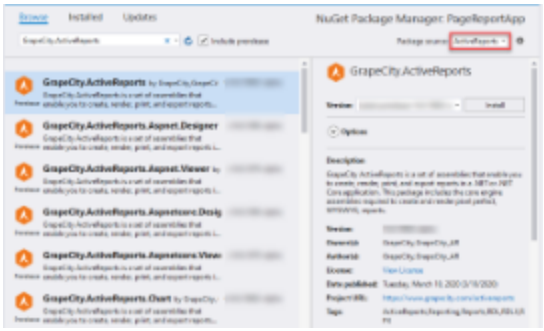
1. Open **NuGet.Config** file placed here: C:\Users\[UserName]\AppData\Roaming\NuGet.
2. Modify the content of NuGet.Config as follows. This adds a key that directs to the path where NuGet packages are available locally.

NuGet.config

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <packageSources>
    <add key="nuget.org" value="https://api.nuget.org/v3/index.json"
protocolVersion="3" />
    <add key="ActiveReports" value="C:\Program Files (x86)\GrapeCity\ActiveReports
14\NuGet" />
  </packageSources>
</configuration>
```

2. Install Packages

1. Open Visual Studio.
2. Create any application (any target that supports .NET Standard 2.0).
3. Right-click the project in Solution Explorer and choose **Manage NuGet Packages**.
4. In the **Package source** on top right, select **ActiveReports** (key added in NuGet.config).
5. Click **Browse** tab on top left and search for 'GrapeCity.ActiveReports'.
6. On the left panel, select **GrapeCity.ActiveReports**.

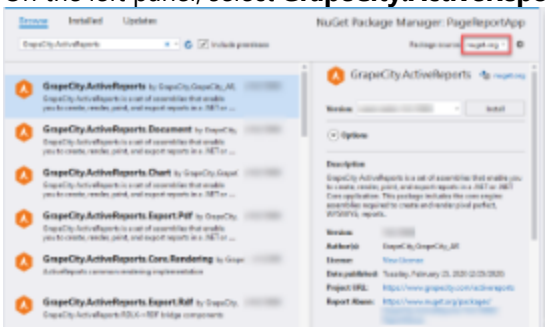


7. On the right panel, click **Install**.
8. In the **License Acceptance** dialog, select **I Accept** to proceed the installation.

Installing Packages from NuGet

GrapeCity ActiveReports 14 references are available through **NuGet** and can be obtained directly from website - https://www.nuget.org/packages?q=GrapeCity.ActiveReports. When you add reference to **GrapeCity.ActiveReports** package, a set of core engine assemblies are added to the application. Use following steps to find and install the NuGet packages in your application:

1. Open Visual Studio.
2. Create any application (any target that supports .NET Standard 2.0).
3. Right-click the project in Solution Explorer and choose **Manage NuGet Packages**.
4. In the **Package source** on top right, select **nuget.org**.
5. Click **Browse** tab on top left and search for 'GrapeCity.ActiveReports'.
6. On the left panel, select **GrapeCity.ActiveReports**.



7. On the right panel, click **Install**.
8. In the **License Acceptance** dialog, select **I Accept** to proceed the installation.

For more information on NuGet configurations, please see Common NuGet configurations and NuGet Configuration Settings articles by Microsoft.

> **Note**: The assemblies are available in the packages at the following location:
>
> - if installer is used: C:\Program Files (x86)\GrapeCity\ActiveReports 14\NuGet\{*Package name*}\lib\net462\{*Assembly*}
> - if a package is installed in an application: [*App name*]\packages\{Package name}\lib\net462\{*Assembly*}

GrapeCity Copyright Notice

Information in this document, including URLs and web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. No part of this document may be reproduced, stored in, or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photo copying, recording, or otherwise), or for any purpose, without the express written permission of GrapeCity, inc.

The ActiveReports License Agreement constitutes written permission for Professional Edition licensees to copy documentation content for distribution with their end user designer applications so long as GrapeCity is given credit within the distributed documentation.

ActiveReports and the ActiveReports logo are registered trademarks of GrapeCity, inc.

All other trademarks are the property of their respective owners. See Open Source Software used in ActiveReports.

End User License Agreement

The End-User license agreement is available online at https://www.grapecity.com/legal/eula.

Please read carefully before installing this software package. Your installation of the package indicates your acceptance of the terms and conditions of this license agreement. Contact GrapeCity Inc. if you have any questions about this license.

Redistributable Files

ActiveReports is developed and published by GrapeCity, Inc. You may use it to develop applications in conjunction with Microsoft Visual Studio or any other programming environment that enables the user to use and integrate the control(s). You may also distribute, free of royalties, the following Redistributable Files with any such application you develop to the extent that they are used separately on a single CPU on the client/workstation side of the network:

- DocumentFormat.OpenXml.dll
- Gcef.Data.DataEngine.dll
- Gcef.DataExpressionInfo.dll
- Gcef.Data.VBFunctionLib.dll
- GrapeCity.ActiveReports.Chart.dll
- GrapeCity.ActiveReports.Aspnetcore.Viewer.dll
- GrapeCity.ActiveReports.Aspnet.Viewer.dll
- GrapeCity.ActiveReports.Aspnetcore.Designer.dl
- GrapeCity.ActiveReports.Aspnet.Designer.dll
- GrapeCity.ActiveReports.Core.DataProviders.dll
- GrapeCity.ActiveReports.Core.Diagnostics.dll
- GrapeCity.ActiveReports.Core.Export.Html.Page.dll
- GrapeCity.ActiveReports.Core.Export.Text.Page.dll
- GrapeCity.ActiveReports.Core.Export.Excel.Page.dll
- GrapeCity.ActiveReports.Core.Drawing.Gdi.dll
- GrapeCity.ActiveReports.Core.Rdl.dll
- GrapeCity.ActiveReports.Core.Rendering.dll
- GrapeCity.ActiveReports.Document.dll

- GrapeCity.ActiveReports.Export.Html.dll
- GrapeCity.ActiveReports.Export.Excel.dll
- GrapeCity.ActiveReports.Export.Image.dll
- GrapeCity.ActiveReports.Export.Pdf.dll
- GrapeCity.ActiveReports.Export.Rdf.dll
- GrapeCity.ActiveReports.Export.Svg.dll
- GrapeCity.ActiveReports.Export.Text.dll
- GrapeCity.ActiveReports.Export.Word.dll
- GrapeCity.ActiveReports.Export.Xml.dll
- GrapeCity.ActiveReports.Imports.Access.dll
- GrapeCity.ActiveReports.Imports.Crystal.dll
- GrapeCity.ActiveReports.Interop.dll
- GrapeCity.ActiveReports.OracleClient.dll
- GrapeCity.ActiveReports.Serializer.dll
- GrapeCity.ActiveReports.dll
- GrapeCity.ActiveReports.Viewer.Win.dll
- GrapeCity.ActiveReports.Viewer.Wpf.dll
- GrapeCity.ActiveReports.VisualStudio.dll
- DefaultWPFViewerTemplates.xaml
- GrapeCity.ActiveReports.Viewer.Html.css
- GrapeCity.ActiveReports.Viewer.Html.js
- GrapeCity.ActiveReports.Viewer.Html.min.js
- jsViewer.min.js
- jsViewer.min.css
- web-designer.js
- web-designer.css
- Newtonsoft.Json.dll
- System.IO.Compression.dll

The following Redistributable Files require the **Professional Edition** license for redistribution:

- GrapeCity.ActiveReports.Design.Win.dll
- GrapeCity.ActiveReports.Web.dll

> **Note**: See Install ActiveReports for the location of the files listed above.

Open Source Software

ActiveReports supports multiple types of Open Source software (hereinafter, referred to as "OSS") programs based on their license agreement and usage rights. OSS is distributed in the hope that it will be useful, but without warranty of any kind, either expressed or implied.

OSS can be used freely for any purpose; however there are terms and conditions for distribution and modification. Please read the Agreement carefully before using it. The following OSS are included as a part of this software.

Open Source Software used in ActiveReports

The OSS's used in ActiveReports according to their usage:

Visual Studio integration

- **Microsoft.VisualStudio.DpiAwareness**
  Copyright (c) Microsoft Corporation
  (Microsoft Visual Studio)
- **VSIXBootstrapper**
  Copyright (c) Microsoft Corporation
  (MIT)
- **NuGet.VisualStudio**
  Copyright (c) Microsoft Corporation
  (Apache-2.0)

Excel and Word Import/Export

- **DocumentFormat.OpenXml**
  Copyright (c) Microsoft Corporation
  (MIT)
- **ColorMine**
  (MIT)

HTML Export

- **SvgNET**
  Copyright (c) 2003 RiskCare Ltd.
  Copyright (c) 2010 SvgNet & SvgGdi Bridge Project
  Copyright (c) 2015-2019 Rafael Teixeira, Mojmír Němeček, Benjamin Peterson and Other Contributors
  (BSD-3-Clause)

PDF Export

- **Portable.BouncyCastle**
  Copyright (c) 2000-2011 The Legion Of The Bouncy Castle
  (MIT)

Data Providers

- **Newtonsoft.Json**
  Copyright (c) 2007 James Newton-King
  (MIT)

Polygon Clipping

- **MartinezClipper**
  Copyright (c) 2016 BrightBit
  (MIT)

FormattedText reader

- **SgmlReader**
  (Apache-2.0)

RPX to RDLX Converter

- **Rtf2Html**
  (Apache-2.0)

RDF document

- **MemoryTributary**
  (CPOL)

End User Designer UI

- **FlagEnumUIEditor**
  (CPOL)

Barcode

- **Okapi Barcode**
  (Apache-2.0)
- **ZXing-CSharp**
  (Apache-2.0)

Query Designer

- **SharpExpress**
  Copyright (c) 2014 Sergey Todyshev
  (MIT)
- **Irony**
  Copyright (c) 2019 Irony Project
  (MIT)
- **jQuery**
  (MIT)
- **Knockout**
  Copyright (c) 2010 Steven Sanderson, the Knockout.js team, and other contributors
  (MIT)
- **Bootstrap**
  Copyright (c) 2011-2020 Twitter, Inc.
  Copyright (c) 2011-2020 The Bootstrap Authors
  (MIT)
- **spin.js**
  Copyright (c) 2011-2018 Felix Gnass
  (MIT)
- **korest**
  Copyright (c) 2014 Sergey Todyshev
  (MIT)

Open Source Software used in JSViewer

JSViewer

- **abortcontroller-polyfill**
  (MIT)
- **classnames**
  Copyright (c) 2017 Jed Watson.
  (MIT)
- **core-js**
  (MIT)

- **downloadjs**
  (MIT)
- **i18next**
  (MIT)
- **i18next-browser-languagedetector**
  (MIT)
- **immutability-helper**
  (MIT)
- **jquery**
  (MIT)
- **moment**
  (MIT)
- **preact**
  (MIT)
- **whatwg-fetch**
  (MIT)

GrapeCity products used in ActiveReports

- GrapeCity.DataVisualization
  Newtonsoft.Json
  Copyright (c) 2007 James Newton-King
  (MIT)
- GrapeCity.Documents.Imaging
  ImageSharp
  (Apache)
- GrapeCity.Documents.Common
- GrapeCity.Documents.Common.Windows
- GrapeCity.Documents.Pdf
  Portable.BouncyCastle
  Copyright (c) 2000-2020 Legion of the Bouncy Castle Inc.
  (MIT)
- GrapeCity.Documents.DX.Windows

GrapeCity products used in JSViewer

- @grapecity/core-ui
  - @grapecity/core-ui-preact
    memoize-one
    (MIT)
    raf
    (MIT)
  - @grapecity/core-ui-react
    deep-object-diff
    (MIT)
    memoize-one
    (MIT)
    raf
    (MIT)

- @grapecity/viewer-core
  moment
  (MIT)

GrapeCity products used in WebDesigner Application

- @grapecity/ar-designer
  - @grapecity/core-ui-react
    deep-object-diff
    (MIT)
    memoize-one
    (MIT)
    raf
    (MIT)
- @grapecity/ar-viewer (JSViewer)
  (same as **previous** section)

License Your ActiveReports

You can use the GrapeCity License Manager utility to license ActiveReports during installation or if you already have a trial version installed. This section gives an overview of all aspects of licensing in ActiveReports.

Topic	Content
License Types	Learn about the types of licenses in ActiveReports.
Licensing a Developer Machine	Learn about licensing a developer machine.
Licensing a Project	Learn about licensing a project.
Licensing Compiled Code	Learn about licensing a compiled code.
Licensing with Pipelines	Learn about licensing with pipelines.
Licensing Errors	Find out solution to common licensing errors.
Contacting Support	Find out how to contact us in case you have problems related to licensing the product.

License Types

See ActiveReports Editions to learn which features are exclusive to the Professional Edition.

License Type	Description
Evaluation	**Trial key is required.** Evaluation banners display on all reports and controls, and the product stops functioning after 30 days from the date of installation. The first key is already activated when you download the trial. If needed, you can request a new key from the Sales department for an additional 30 day trial.
Standard	**Standard Edition product key is required.** Evaluation banners appear only on features that are exclusive to the Professional Edition. You receive this key by email when you purchase ActiveReports Standard Edition or upgrade from a previous version of ActiveReports Standard

	Edition.
Professional	**Professional Edition product key is required.** All reporting functionality and controls appear without any evaluation banners. You receive this key by email when you purchase ActiveReports Professional Edition or upgrade from a previous version of ActiveReports Professional Edition.

If you cannot find your email with the product key, please contact activereports.sales@grapecity.com to have it looked up.

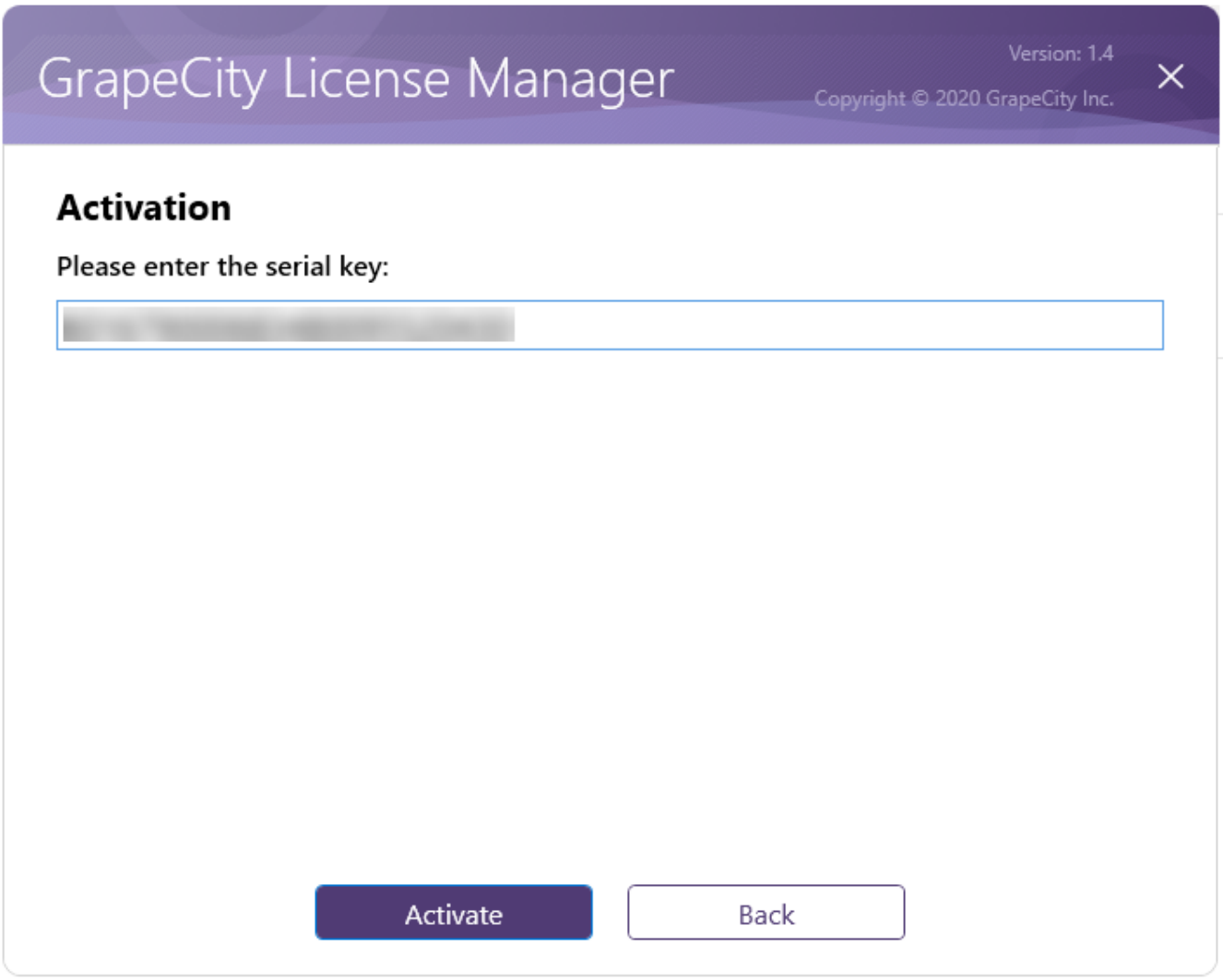
Licensing a Developer Machine

You need to license the machine, on which, you open the ActiveReports or compile the ActiveReports projects in Visual Studio.

> **Tip:** Always run the License Manager as an **Administrator**.

To license a machine with ActiveReports during installation or to license a trial without reinstalling

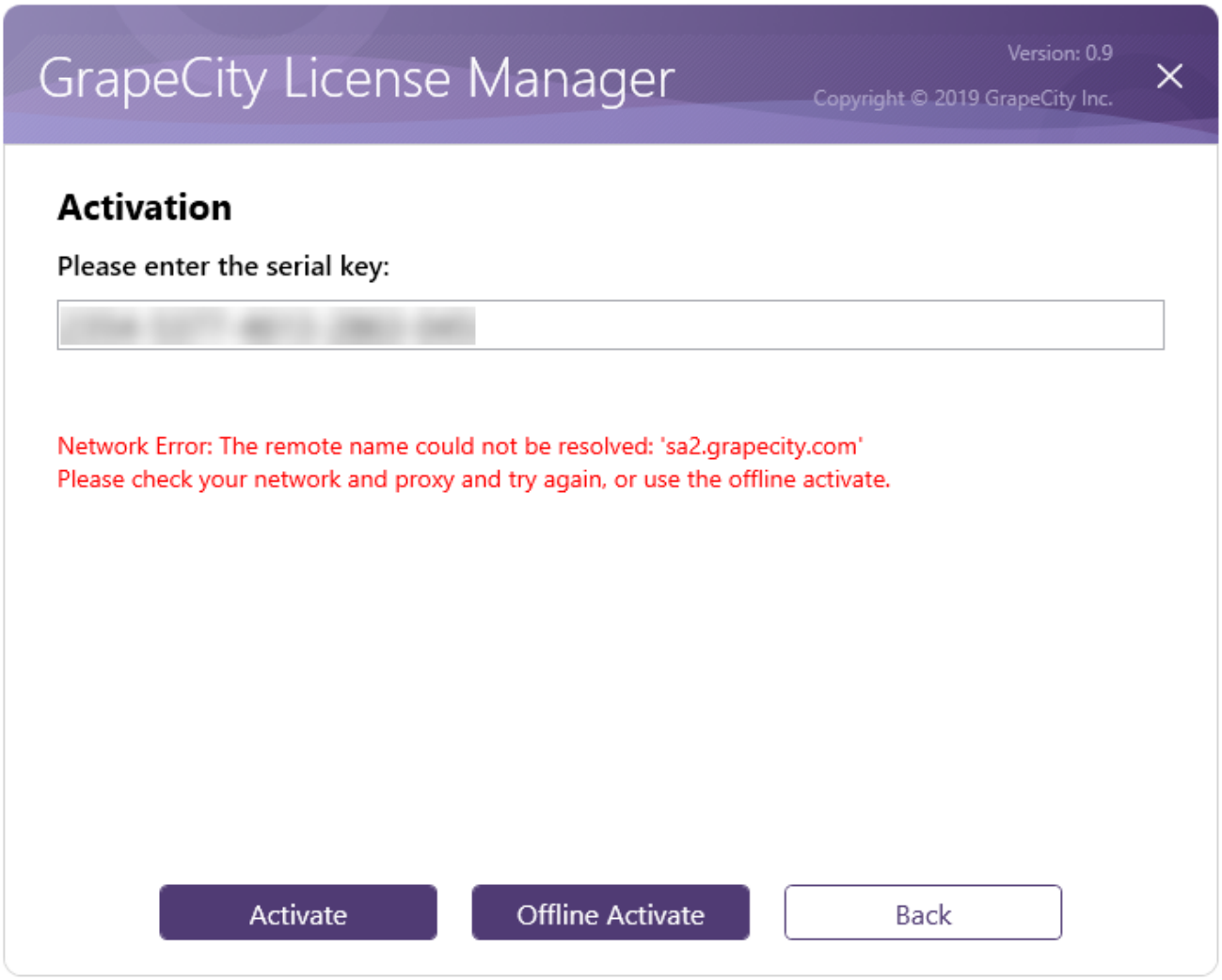
1. Go to the Start menu and select **GrapeCity License Manager** to open the licensing manager window. This window appears automatically during the product installation.
2. In the **GrapeCity License Manager** window, click **Activate**.
3. Enter the serial key, you have received from GrapeCity (including all the capital letters and special characters), and click **Activate**.



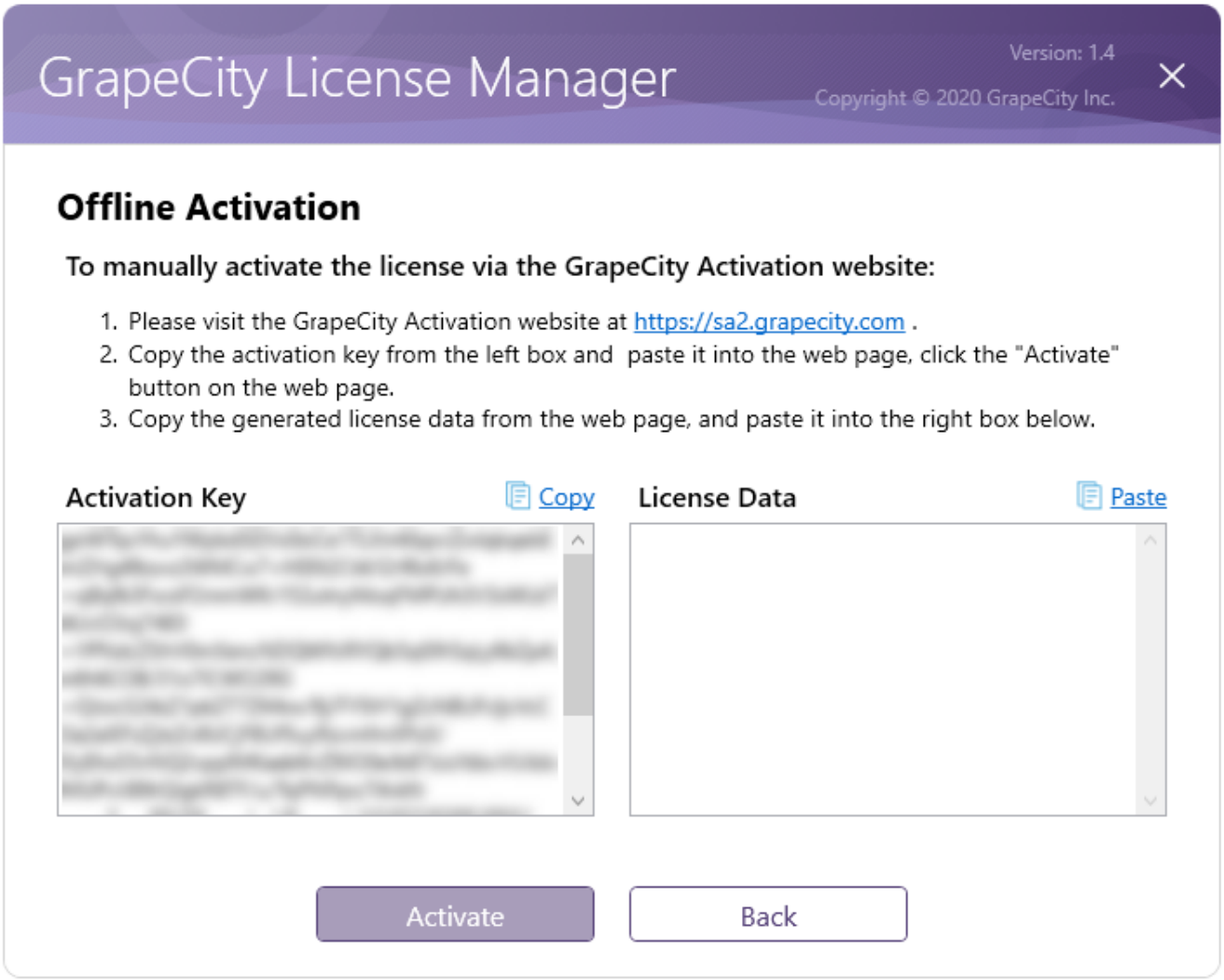
4. An activation message appears on the window. Click **OK** to complete the licensing process.
   To see the license details such as the license type (Trial or Product), edition (Professional or Standard), and the date of activation and expiry, click **Details** in the licensing window.

To license ActiveReports on a machine without an internet connection

1. Go to the Start menu and select **GrapeCity License Manager** to open the licensing manager window. This window appears automatically during the product installation.
2. In the **GrapeCity License Manager** window, if you try to enter the serial key and click **Activate**, a network error is shown on the window. So, in such a case, click **Offline Activate** to proceed further with the licensing process.

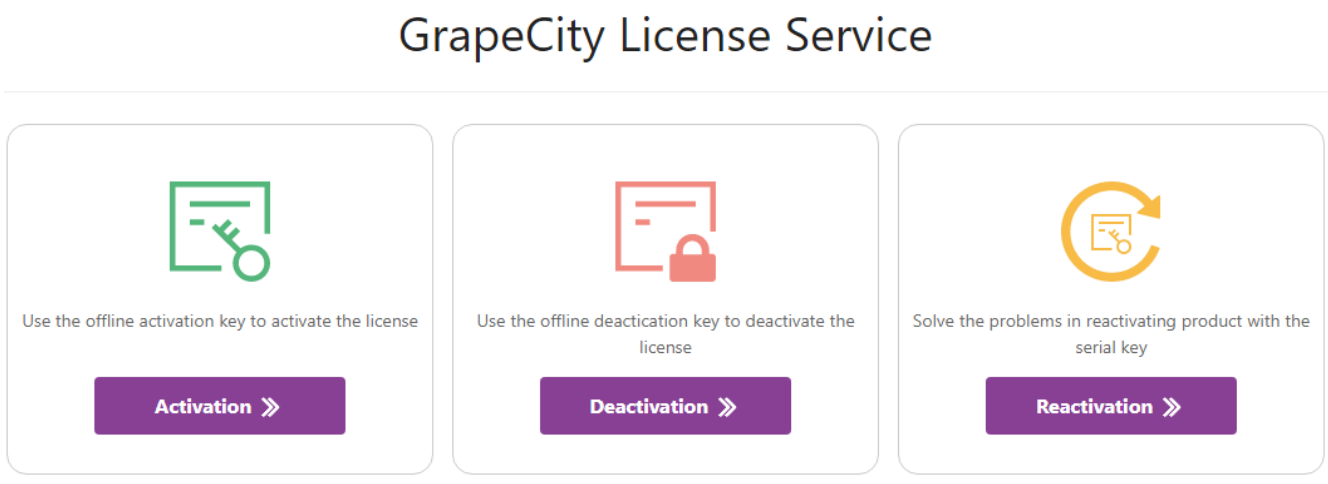


3. Copy the Activation Key from the **GrapeCity License Manager** window.

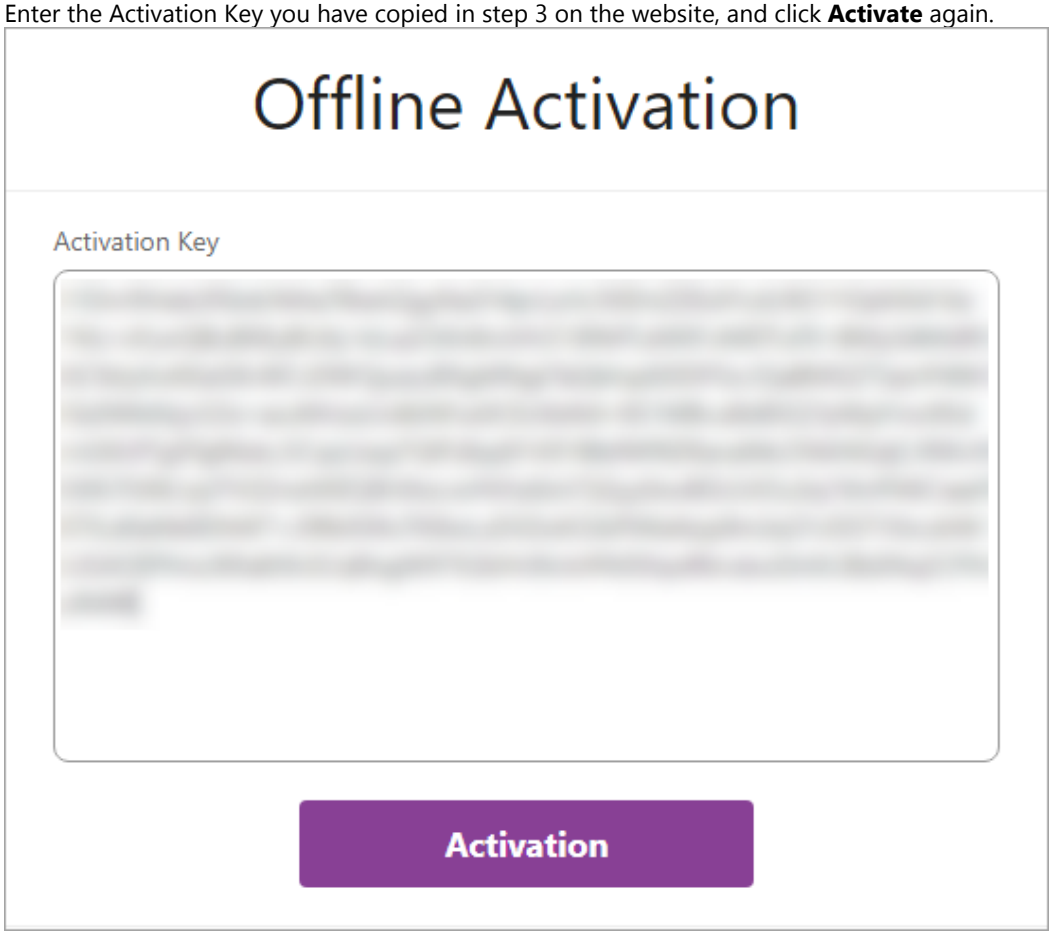


4. Visit the following website https://sa2.grapecity.com on the other machine (having an internet connection), and click **Activation** to use offline Activation key to activate the license.

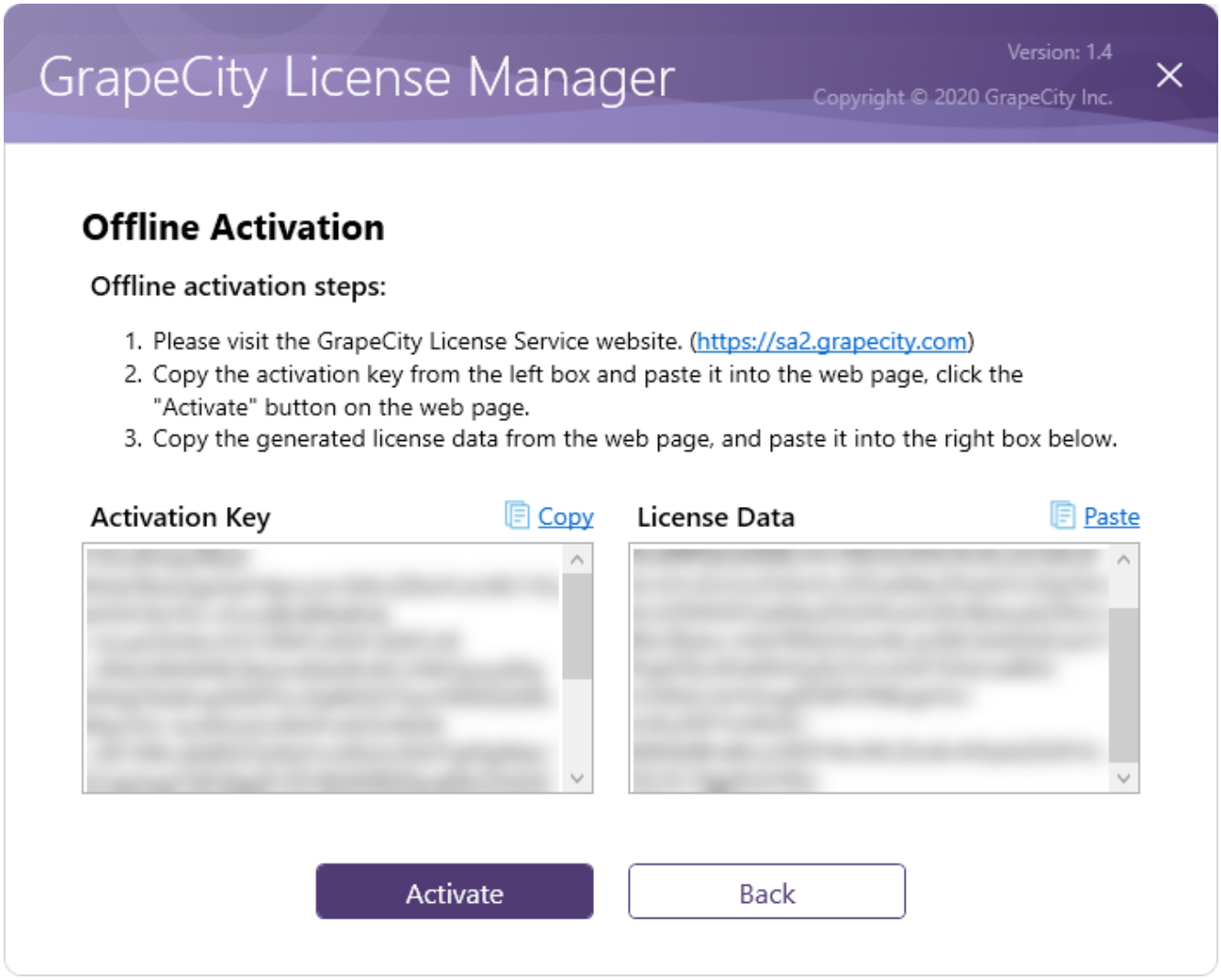
> **Note:** Do not close the activation dialog on your original machine until the activation process is completed.



5. Enter the Activation Key you have copied in step 3 on the website, and click **Activate** again.



6. After a successful activation on the website, copy the License Data from the website and paste it in the licensing window.



7. Click **Activate** and then **OK** to complete the licensing process.

To activate a license for ActiveReports on multiple machines

You can activate a single developer license key for ActiveReports on **three machines** for use by **one developer**.

If you have used all the three license activations and you wish to license a fourth machine, for example, a virtual machine for use by the same licensed developer, then you must deactivate the license key from one of the three machines. For more information on this, see the section **"To deactivate an ActiveReports license."**

After you have deactivated licensing on one of the machines, you can then activate ActiveReports on the other machine.

> **Note**: Deactivate your ActiveReports license before formatting a machine to avoid loss of activation. If this happens by accident, you can contact our support team for help.

To deactivate an ActiveReports license

When you have used all the three of your license activations and you still want to license an another machine for your own use. So, in such a case you can deactivate an ActiveReports license on any of three machines to license the fourth machine.

Follows these steps to deactivate an ActiveReports license on a machine:

1. Go to the Start menu and select **GrapeCity License Manager** to open the licensing manager window.
2. In the **GrapeCity License Manager** window, click **Deactivate**.
3. In the **Deactivation** page, click **Deactivate** again to confirm the deactivation process.
4. Click **OK** to complete the deactivation process.

To upgrade or downgrade a license

> **Note:** In order to upgrade or downgrade a license on a machine, you need to install both the Professional and Standard licenses on the machine. If you have not, then the Upgrade/Downgrade column does not appear in the **GrapeCity License Manager** window. And, in case, you have only one of the licenses and you wish to install the other, then you must first deactivate the installed license. Once you deactivate the license, the Upgrade/Downgrade column is automatically added at that point.

Follow these steps to change your ActiveReports license type:

Upgrade from a Standard to a Professional License:

1. Go to the Start menu and select **GrapeCity License Manager** to open the licensing manager window.
2. In the **GrapeCity License Manager** window, click **Upgrade to Professional License** under the Upgrade/Downgrade column.
3. Follow the activation steps from Step 3 onwards of **To license an ActiveReports Trial without reinstalling** to upgrade.

Downgrade from a Professional to a Standard License:

1. Go to the Start menu and select **GrapeCity License Manager** to open the licensing manager window.
2. In the **GrapeCity License Manager** window, click **Downgrade to Standard License** under the Upgrade/Downgrade column.
3. In the **Deactivate ActiveReports 14** page, click the **Next** button.
4. In the **Confirm the Product** screen, confirm that the correct product is being downgraded and click the **Next** button.
5. After a successful completion, a **Deactivation Successful** screen appears with the **Product Name** as ActiveReports 14 and the **Current Status** as Standard License.

> **Note:** You can upgrade a license only if you buy an upgrade from Standard Edition to Professional Edition.

To determine whether a machine is licensed

1. Open any sample in Visual Studio from the included samples located in C:\Users**USERNAME**\Documents\GrapeCity Samples\ActiveReports 14 and click the **Preview** tab.
2. Scroll to the bottom of the report and check for any red evaluation text. If there is any, it means your machine is not licensed.

> **Note:** To check for Professional Edition licensing, open a sample from the **Professional** folder, run it, and look for the evaluation messages.

Licensing a Project

> **Tips:**
>
> - To deploy using XCOPY, you must include the DLLs for all of your ActiveReports references in your bin/debug folder. To do this, in the Visual Studio Solution Explorer, select each reference, and in the Properties window, set **Copy Local** to **True** and rebuild your solution.
>
> - To avoid having to change the version number each time you install an ActiveReports service pack. You need

> to select each reference in the Visual Studio Solution Explorer, and in the Properties window, set **Specific Version** to **False**.

To license Windows Forms projects made on the trial version

These steps assume that you have an ActiveReports licensed edition installed on your system.

1. Open the project in Microsoft Visual Studio.

   > **Note:** If another application calls the one containing ActiveReports features, you must license the calling application to avoid evaluation banners after deployment.

2. In Visual Studio, go to the **Build** menu and select **Rebuild Solution**.
3. To verify that the application is licensed, open the licenses.licx file and compare it to the Required references section below.

The executable application is now licensed, and no nag screens or evaluation banners will appear when you run it. You can also distribute the application to the unlicensed machines without facing any nag screens or evaluation banners.

Required references in the licenses.licx file (for Standard and Professional Editions)

The licenses.licx file must contain the following references to ActiveReports if you are using both the Section and Page reports, and both the Windows and WPF viewers. See the table for the references to the ActiveReports version and the reference to the Viewer control.

> **Note:** The Version, Culture, and PublicKeyToken information is added automatically, but they can be removed, and are preferred to be removed, if the version is wrong.

Paste INSIDE the licenses.licx file.

```
GrapeCity.ActiveReports.SectionReport, GrapeCity.ActiveReports
GrapeCity.ActiveReports.PageReport, GrapeCity.ActiveReports
GrapeCity.ActiveReports.Viewer.Win.Viewer, GrapeCity.ActiveReports.Viewer.Win
GrapeCity.ActiveReports.Viewer.Wpf.Viewer, GrapeCity.ActiveReports.Viewer.Wpf
```

The below table lists all the license strings that you may need:

Component	License String
Section report engine	GrapeCity.ActiveReports.SectionReport, GrapeCity.ActiveReports
Page and RDL report engine	GrapeCity.ActiveReports.PageReport, GrapeCity.ActiveReports
WinForms viewer control	GrapeCity.ActiveReports.Viewer.Win.Viewer, GrapeCity.ActiveReports.Viewer.Win
WPF viewer control	GrapeCity.ActiveReports.Viewer.Wpf.Viewer, GrapeCity.ActiveReports.Viewer.Wpf
PRO (some features) PDF export	GrapeCity.ActiveReports.Export.Pdf.Section.PdfExport, GrapeCity.ActiveReports.Export.Pdf
PRO ONLY: WebViewer, HTTP handlers	GrapeCity.ActiveReports.Web.WebViewer, GrapeCity.ActiveReports.Web
PRO ONLY: End-user	GrapeCity.ActiveReports.Design.Designer, GrapeCity.ActiveReports.Design.Win

designer	
PRO ONLY: JSViewer	GrapeCity.ActiveReports.Aspnet.WebViewer, GrapeCity.ActiveReports.Aspnet.Viewer
PRO ONLY: Web Designer	GrapeCity.ActiveReports.Aspnet.WebDesigner, GrapeCity.ActiveReports.Aspnet.Designer

Note: When using the PDF export filter in your project, make sure you check the licenses.licx file for reference to the PDF Export Assembly.

To license Web Forms projects made on the trial version

Follow these steps after you have licensed ActiveReports on your machine:

1. Open the Web Forms project in Microsoft Visual Studio.
2. In Visual Studio, go to the **Build** menu and select **Rebuild Solution**.
3. The web site application is now licensed. You can distribute the web site application to unlicensed machines and no evaluation banners will appear.

To license Web Site applications

Follow these steps after you license ActiveReports on your machine.

1. Open the project in Visual Studio.
2. In the Solution Explorer, right-click the licenses.licx file and select **Build Runtime Licenses** to create the App_Licenses.dll file.
3. The web site application is now licensed. You can distribute the web site application to unlicensed machines without facing any evaluation banners.

To license a class library project

Follow these steps after you have licensed ActiveReports on your machine.

1. Open the project for the root-level calling application, that is, the executable that calls your class library, in Visual Studio.
2. From the **Project** menu, select **Add New Item**, and select an ActiveReports report. (You can delete it later. This is only to add the references to the project.)
3. In Visual Studio, go to the **Build** menu and select **Rebuild Solution**.
4. Check the licenses.licx file and verify that ActiveReports licensing is added for all of the features used in your project.

Note:

- If you use some other features of ActiveReports in your class library that are still showing an evaluation banner, for example, features exclusive to the Professional Edition, you can add those references manually and rebuild the solution.
- The **Web Key Generator** and **Application License Generator** utilities have been removed in ActiveReports 14. You should use above steps to license the project.

To license a project that use Web Designer and JSViewer nuget packages

The Web Designer and JSViewer nuget packages do not include a licenses.licx file, so the file needs to be added and configured manually in your project. Follow these steps to add license file in your Web Application in Visual Studio:

1. Go to **Project** menu and select **Add New Item**.
2. Select a new text file and rename it to **licenses.licx**.
3. Open the blank licenses.licx file and populate the file with the following entries:

Paste INSIDE the licenses.licx file.

`GrapeCity.ActiveReports.SectionReport,`
`GrapeCity.ActiveReportsGrapeCity.ActiveReports.PageReport,`
`GrapeCity.ActiveReportsGrapeCity.ActiveReports.Export.Pdf.Section.PdfExport,`
`GrapeCity.ActiveReports.Export.Pdf`

4. Run the project.
   You should not face any licensing errors, assuming that you have a licensed ActiveReports Professional Edition.

Licensing Compiled Code

This section describes how to generate the license for the compiled code or specifically for the applications deployed on Azure Functions application. By default, licensing is applied to your main application. In case ActiveReports is embedded in a custom library, that is called by another application, a "license not found" error is shown. Therefore, you need to generate a license file for the target application.

For example, there is a 'UserControlLibrary' project with ActiveReports libraries embedded and a 'MainApp' project that references the 'UserControlLibrary'. To generate license for the specified target application, follow these steps:

1. Open the command line on Windows and change the working directory to:
   `C:\ProgramData\GrapeCity\gclm`

2. Run the command as:
   `gclm.exe "de456e2c-d2e3-4246-94e8-9648bbd6fbf0"  -lc "[output dir].gclicx [entry assembly name].[calling assembly name].dll"`

   For example,
   `C:\ProgramData\GrapeCity\gclm>gclm.exe "de456e2c-d2e3-4246-94e8-9648bbd6fbf0" -lc ./.gclicx "MainApp.UserControlLibrary.dll"`

   To license an application on an **Azure Functions** application, run the following command:
   `gclm.exe "de456e2c-d2e3-4246-94e8-9648bbd6fbf0" -lc .\.gclicx Microsoft.Azure.WebJobs.Script.WebHost.[assembly name].dll`

   It generates a **.gclicx** file in the output directory.

   > **Note:** You must specify the application name while generating a license. Also, the generated license can not be used in other applications with different names.

3. Copy the **.gclicx** file and paste it into your application (in this case, UserControlLibrary) where ActiveReports assemblies are used, in any folder.
4. Change the build action of **.gclicx** to **Embedded Resource**.
5. Rebuild the solution and run the project.
   Our control will now be able to lookup license for the 'MainApp' from the 'UserControlLibrary' assembly.

   In case of licensing the application on **Azure Functions** application, deploy the application on Azure Function.

⚠
- This licensing process requires an internet connection.
- For the license to apply at runtime, you must call **gclm.exe** with the target name, for example, 'AppName.PluginLibraryName.dll'.
- The build action of **.gclicx** file in the project must set to **Embedded Resource**.

Licensing with Pipelines

The **GrapeCity License Manager** (gclm) tool identifies the machine via the host-name. However, in pipeline environments, the build actions work in virtual machines or dockers on cloud, so the host-name would be different in each build process. The environments may also be reset before every build so that the activated license is removed.

The steps to license the applications with pipelines are as follows:

1. Deploy the application. If the **gclm** tool is not already installed, it will be installed by NuGet packages if the pipeline has access to it. The **gclm** tool is also available on CDN here: https://cdn.grapecity.com/license/gclm_deploy.exe Note that the project is built with **Trial License** for 30 days.
2. Activate the serial key. Run the following command in the command line:
   `gclm.exe "de456e2c-d2e3-4246-94e8-9648bbd6fbf0" -a [serial key]`
3. Rebuild the application with the activated license. This step is required since the gclm tool is run and the license is validated.
4. Run the deployed application using dotnet command. For example:
   `dotnet JSViewerApplication.dll`
5. Deactivate the license activated in Step#2. Run the following command in the command line:
   `gclm.exe "de456e2c-d2e3-4246-94e8-9648bbd6fbf0" -d [serial key]`

   The license should be activated.

Licensing Errors

Here are some common licensing errors and their causes.

Error	Cause
Application cannot run because it was built with no license.	Licensing is not present in the application or the calling application. See below for information on how to license the calling application.
License for XXXX (control name) could not be found.	Extra lines for components that you do not use are in the licenses.licx file. Delete unnecessary information and Rebuild the project.
Licensing has not been correctly applied to the application.	Check the three key points below.
Exception (LicenseException)	Check the three key points below.
'No License' message on running **.NET Core**	This is so because Visual Studio 2017 does not support core license compilation.

applications in Visual Studio 2017	

1. **Ensure that the license file is added to the appropriate project.**

   The licenses.licx file is automatically generated in the project where ActiveReports is used. But if your application is composed of multiple projects and another project calls the reports defined in your class library, you need to register it in the calling project rather than just in your report project.

   When you add an ActiveReports web service to a Page report, RDL report, or XML-based Section report project, the licenses.licx file is not created automatically, and the license strings are not added. You also need to manually add licensing to your application if you want to create a control at run time or use the HTTP handlers.

   To manually add licensing to the calling application

   > **Note:** In a C# Windows Forms project, the license file is in the **Properties** folder. In a Visual Basic project, it is in the **My Project** folder.

   1. In your application that contains ActiveReports components, check that the proper licensing strings are in the licenses.licx file. (See the table of license strings below.)
   2. Copy the ActiveReports strings from the file into the licenses.licx file in the calling application.
      a. If there is no license file, from the **Project** menu, select **Add New Item**.
      b. From the listed templates, select **Text** file and change the name to "**licenses.licx.**"
      c. In the Solution Explorer, double-click the newly created licenses.licx file to open it, and paste in the licensing strings for all components you use.
   3. From the **Build** menu, select Rebuild Project to embed the licensing.

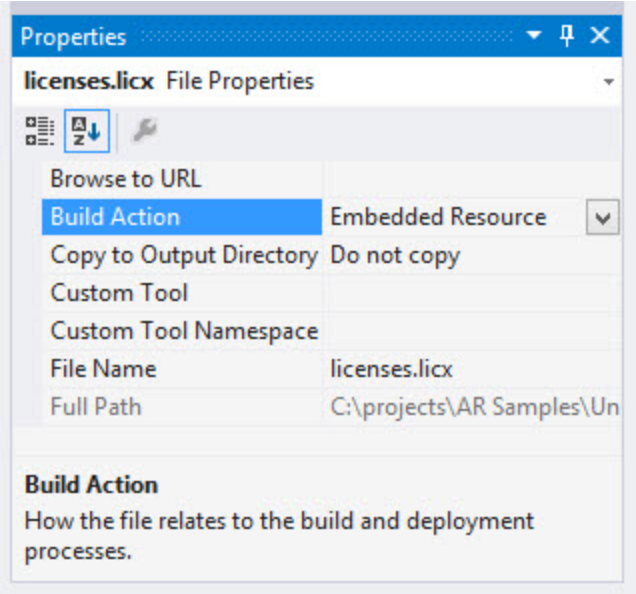
   > **Note:** If your project is a web site, the bin folder has the licenses embedded in the App_Licenses.dll file.

2. **Ensure that the contents of the license file are correct.**

   Depending on which features of ActiveReports you use in your application, the license file may need to contain multiple license strings.

   You will find a full list of license strings that you may need in the **Required references in the licenses.licx file (for Standard and Professional Editions)** section above.

3. **Ensure that the Build Action property is configured correctly for the license file.**



1. In the Solution Explorer, select licenses.licx (you may need to click the **Show all files** button to see it).
2. In the Properties window, ensure that the **Build Action** property is set to **Embedded Resource**.

4. **Ensure that .NET Core applications are always run on Visual Studio 2019**

   .NET Core applications should be run only in **Visual Studio 2019** since Visual Studio 2017 does not support core license compilation.

Contacting Support

If you still face problems licensing ActiveReports, please contact our **Support Team** using any of these methods:

World Wide Web site	https://www.grapecity.com/support/contact
E-mail	activereports.sales@grapecity.com
Phone[1]	(412) 681-4738

[1]Phone support is available for customers that have **Platinum Support** included with their purchased products or **issues** related to licensing or installation of a product. Monday through Friday during regular business hours (EST).

For information on Platinum Support, visit our website at https://www.grapecity.com/support/plans/.

Quick Start

Quickly begin using ActiveReports by following the steps of some of the most commonly used features. See Install ActiveReports topic to see how to add references to GrapeCity.ActiveReports from **nuget.org**.

Add ActiveReports Controls to Toolbox

You can **add an ActiveReport to a project** without using the Visual Studio toolbox, but in order to use the Viewer control, any of the exports, the Designer and related controls, or the WebViewer control, you need to have them in your toolbox.

On installing NuGet packages, the controls are automatically added to the toolbox in Visual Studio 2017 and Visual Studio 2019, in an **ActiveReports 14** tab.

Visual Studio 2013 and Visual Studio 2015

If you are using Visual Studio 2013 or Visual Studio 2015, the NuGet packages do not add the controls on the toolbox by default. You need to add the controls manually as follows.

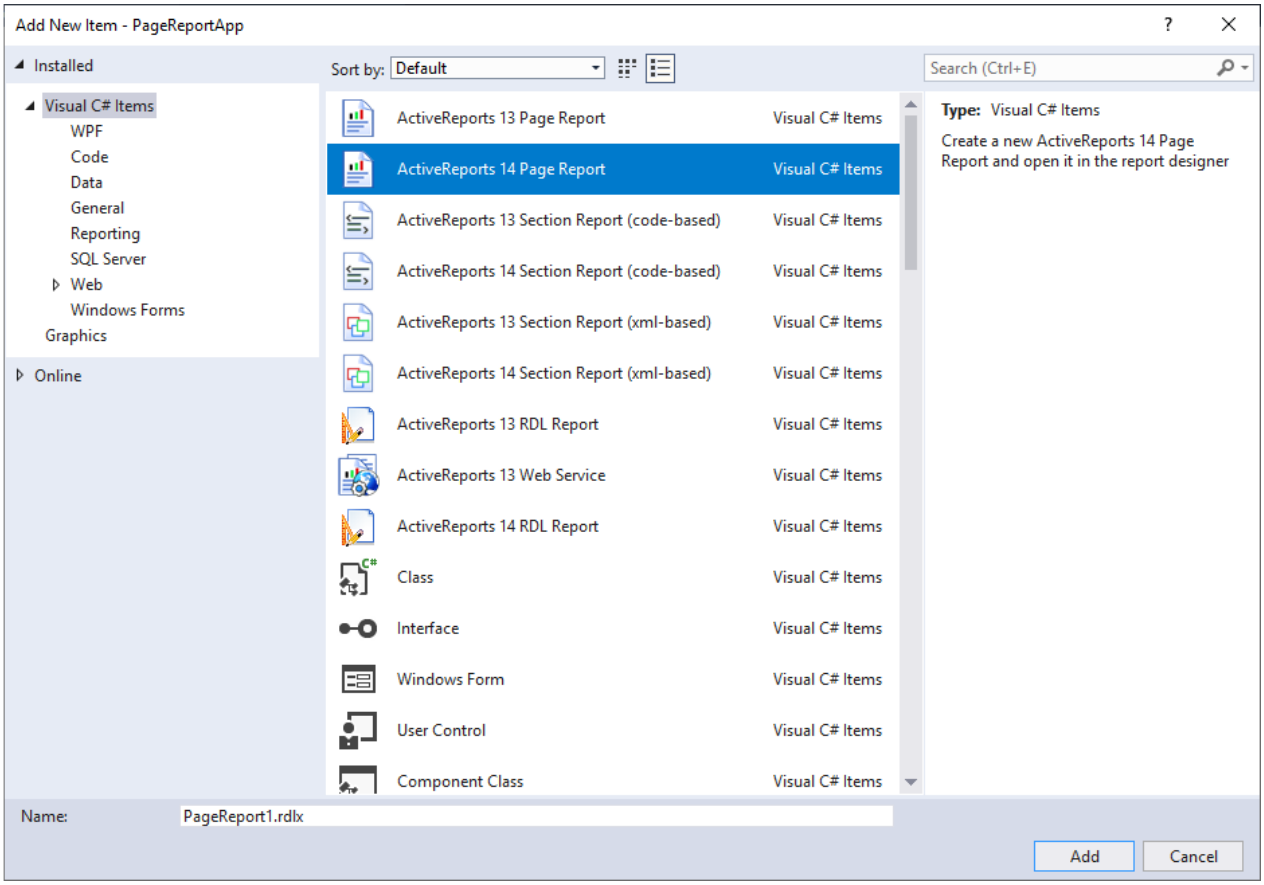
1. Install the NuGet packages (in WinForms or WebForms application).
2. Right-click anywhere on the toolbox and select **Add Tab**.
3. Right-click the newly added tab and select **Choose Items**.
4. In the Choose Toolbox Items dialog with .NET Framework Components tab selected, click **Browse**.
5. Select the assembly present inside the package you installed in the application. The path to the assembly is similar to the following:
   [*App name*]\packages\{*Package name*}\lib\net462\{*Assembly*}

The added control(s) should now display in the tab.

> ⚠ **Limitation**: Items for ActiveReports 14 cannot be added if the toolbox already contains items for ActiveReports 13. This is because assembly names in both ActiveReports versions are the same. To resolve this issue, remove the toolbox items for ActiveReports 13, and then add the items for ActiveReports 14.

Add an ActiveReport to a project

1. From the Visual Studio **Project** menu (or **Website** menu in Web projects), select **Add New Item**.
2. Select the type of report that you want to add (for information on the differences, see Report Types):
   - Section Report (code-based)
   - Section Report (xml-based)
   - Page Report
   - RDL Report



3. In the **Name** box, type a name for the report, and click **Add.** The selected report type is added to your project and opens in the report designer.

Add an ActiveReport to a project at run time

1. In Visual Studio, create a new **Windows Forms Application** or open an existing one.
2. From the Visual Studio toolbox, drag the Viewer control onto your Windows Form.
3. Set the Viewer's **Dock** property to **Fill** to show the complete Viewer control on the form.
4. On the Form.cs or Form.vb that opens, double-click the title bar to create the Form_Load event.
5. Add the following code inside the Form_Load event.

Page Report

Visual Basic.NET code. Paste INSIDE the Form Load event.

```
' Create a new Page report instance
Dim pageReport As New GrapeCity.ActiveReports.PageReport()
Dim Page As New GrapeCity.ActiveReports.PageReportModel.Page()
Dim fixedPage As New GrapeCity.ActiveReports.PageReportModel.FixedPage()

' Add a textbox to your Page report
Dim textbox1 As New GrapeCity.ActiveReports.PageReportModel.TextBox()
textbox1.Name = "TextBox1"
textbox1.Height = "1cm"
textbox1.Width = "10cm"
textbox1.Left = "2cm"
textbox1.Top = "0.5cm"
```

```
textbox1.Value = "Sample Page Report"
Page.ReportItems.Add(textbox1)
fixedPage.Pages.Add(Page)
pageReport.Report.Body.ReportItems.Add(fixedPage)

' Create a Page document and load it in Viewer
Dim pageDocument As New GrapeCity.ActiveReports.Document.PageDocument(pageReport)
viewer1.LoadDocument(pageDocument)
```

C# code. Paste INSIDE the Form Load event.

```
// Create a new Page report instance
GrapeCity.ActiveReports.PageReport pageReport = new GrapeCity.ActiveReports.PageReport();
GrapeCity.ActiveReports.PageReportModel.Page Page = new
GrapeCity.ActiveReports.PageReportModel.Page();
GrapeCity.ActiveReports.PageReportModel.FixedPage fixedPage = new
GrapeCity.ActiveReports.PageReportModel.FixedPage();

// Add a textbox to your Page report
GrapeCity.ActiveReports.PageReportModel.TextBox textbox1 = new
GrapeCity.ActiveReports.PageReportModel.TextBox();
textbox1.Name = "TextBox1";
textbox1.Height = "1cm";
textbox1.Width = "10cm";
textbox1.Left = "2cm";
textbox1.Top = "0.5cm";
textbox1.Value = "Sample Page Report";
Page.ReportItems.Add(textbox1);
fixedPage.Pages.Add(Page);
pageReport.Report.Body.ReportItems.Add(fixedPage);

// Create a Page document and load it in Viewer
GrapeCity.ActiveReports.Document.PageDocument pageDocument = new
GrapeCity.ActiveReports.Document.PageDocument(pageReport);
viewer1.LoadDocument(pageDocument);
```

RDL Report

Visual Basic.NET code. Paste INSIDE the Form Load event.

```
' Create a new RDL report instance
Dim rdlReport As New GrapeCity.ActiveReports.PageReport()

' Add a textbox to your RDL report
Dim textbox1 As New GrapeCity.ActiveReports.PageReportModel.TextBox()
textbox1.Name = "TextBox1"
textbox1.Height = "1cm"
textbox1.Width = "10cm"
textbox1.Left = "2cm"
textbox1.Top = "0.5cm"
textbox1.Value = "Sample RDL Report"
rdlReport.Report.Body.ReportItems.Add(textbox1)

' Create a Page document and load it in Viewer
Dim rdlDocument As New GrapeCity.ActiveReports.Document.PageDocument(rdlReport)
```

```
viewer1.LoadDocument(rdlDocument)
```

C# code. Paste INSIDE the Form Load event.

```
// Create a new RDL report instance
GrapeCity.ActiveReports.PageReport rdlReport = new GrapeCity.ActiveReports.PageReport();

// Add a textbox to your RDL report
GrapeCity.ActiveReports.PageReportModel.TextBox textbox1 = new
GrapeCity.ActiveReports.PageReportModel.TextBox();
textbox1.Name = "TextBox1";
textbox1.Height = "1cm";
textbox1.Width = "10cm";
textbox1.Left = "2cm";
textbox1.Top = "0.5cm";
textbox1.Value = "Sample RDL Report";
rdlReport.Report.Body.ReportItems.Add(textbox1);

// Create a Page document and load it in Viewer
GrapeCity.ActiveReports.Document.PageDocument rdlDocument = new
GrapeCity.ActiveReports.Document.PageDocument(rdlReport);
viewer1.LoadDocument(rdlDocument);
```

Section Report

Visual Basic.NET code. Paste INSIDE the Form Load event.

```
' Create a new Section report instance
Dim sectionReport As New GrapeCity.ActiveReports.SectionReport()

' Create a Detail section
sectionReport.Sections.Add(GrapeCity.ActiveReports.Document.Section.SectionType.Detail, "Body")

' Add a textbox to your Section report
Dim textbox1 As New GrapeCity.ActiveReports.SectionReportModel.TextBox()
textbox1.Name = "TextBox1"
textbox1.Height = 1.5F
textbox1.Width = 10.0F
textbox1.Left = 0.5F
textbox1.Top = 0.5F
textbox1.Value = "Sample Section Report"
sectionReport.Sections(0).Controls.Add(textbox1)

' Load the Section report in the Viewer
sectionReport.Run()
viewer1.LoadDocument(sectionReport)
```

C# code. Paste INSIDE the Form Load event.

```
// Create a new Section report instance
GrapeCity.ActiveReports.SectionReport sectionReport = new GrapeCity.ActiveReports.SectionReport();

// Create a Detail section
sectionReport.Sections.Add(GrapeCity.ActiveReports.Document.Section.SectionType.Detail, "Body");

// Add a textbox to your Section report
```

```
GrapeCity.ActiveReports.SectionReportModel.TextBox textbox1 = new
GrapeCity.ActiveReports.SectionReportModel.TextBox();
textbox1.Name = "TextBox1";
textbox1.Height = 1.5F;
textbox1.Width = 10F;
textbox1.Left = 0.5F;
textbox1.Top = 0.5F;
textbox1.Value = "Sample Section Report";
sectionReport.Sections[0].Controls.Add(textbox1);

// Load the Section report in the Viewer
sectionReport.Run();
viewer1.LoadDocument(sectionReport);
```

Add a Data Source to a Report

The first thing you probably want to do when you create a report is to add data. You can accomplish this in a variety of ways, depending on the type of report you are using.

Page Report/RDL Report Data

With page reports or RDL reports, you basically connect to a data source, and then add a dataset. You can also create a shared data source if you use the same one for many reports. For information on how to perform these tasks, see Work with Data in the How To section. For more information on each item in the associated dialogs, see Data Sources and Datasets in the Concepts section.

For more advanced ways to connect data, see the Walkthroughs section for step by step instructions on using Reports with Stored Procedures, or creating a Custom Data Provider.

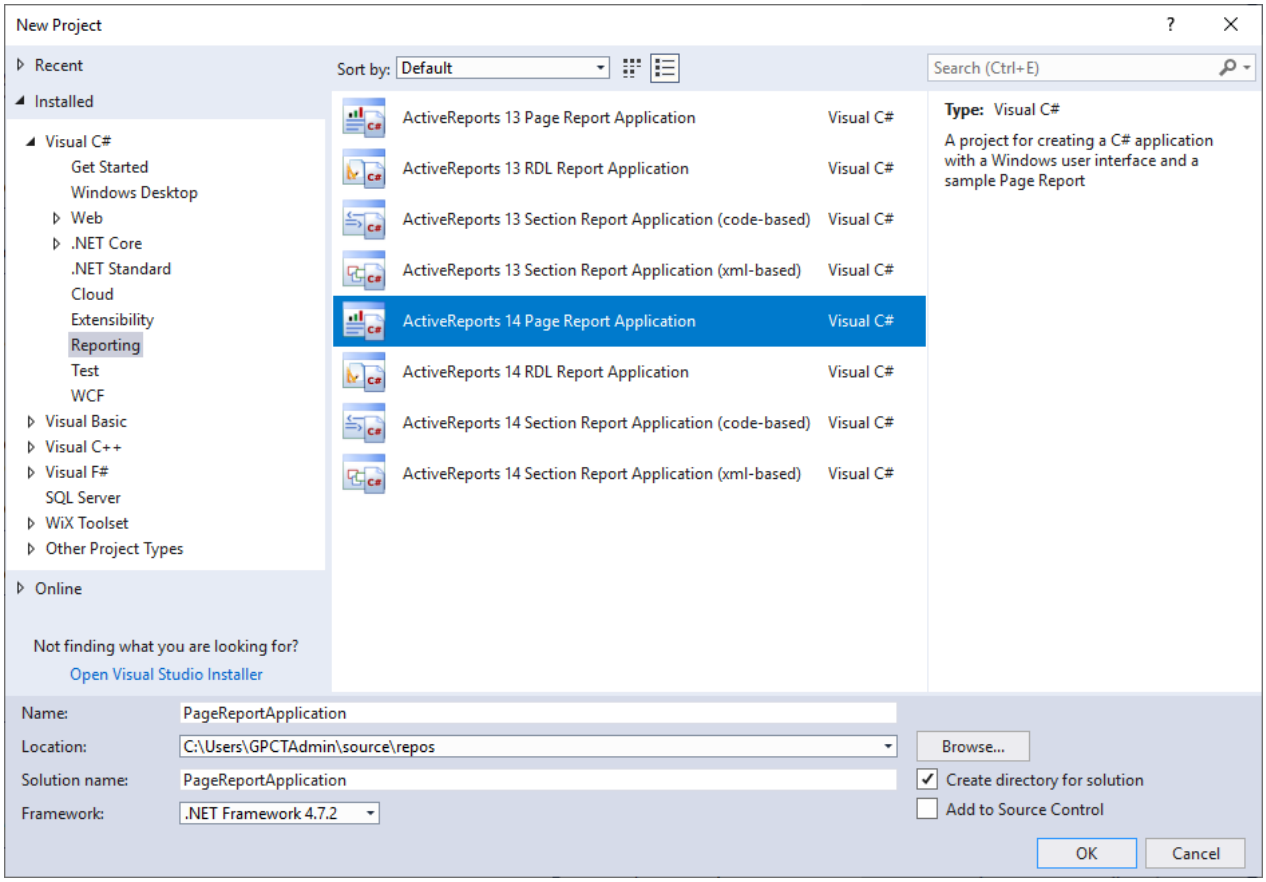
Section Report Data

With section reports, you bind a report to any of a variety of data sources and select the data using a SQL query or XPath expression in the Data Source Dialog. You can also use code to create an unbound data source or to change the data source at run time. For more information on all of these methods of binding reports to data, see Work with Data in the Section Report How To section.

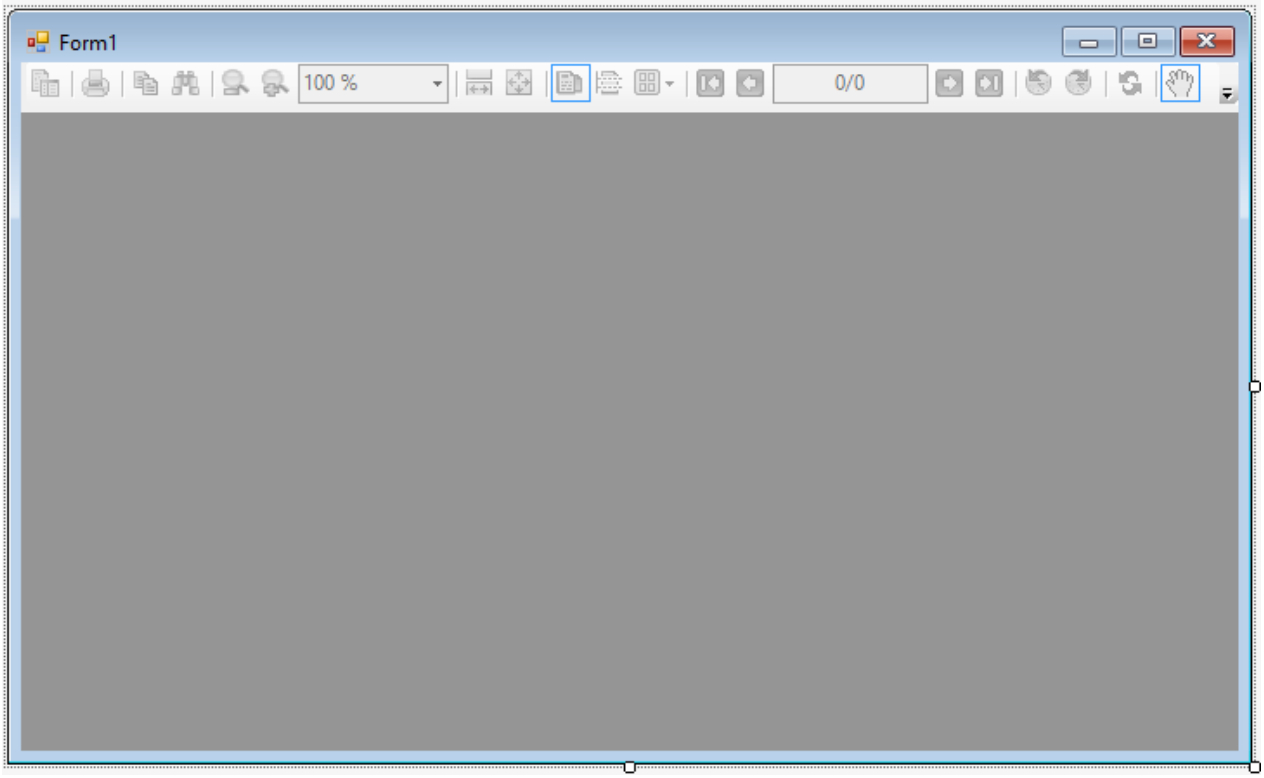
Add an ActiveReports application to a Project

ActiveReports provides an in-built sample application that includes a report template along with the Viewer control. You learnt about creating a report and viewing it in the preceding sections. With this Windows Forms application you only need to create a report layout and run the application to view the report, effectively skipping the manual process of adding a Viewer and template separately and creating an instance of the report.

1. From the Visual Studio **File** menu, select **New**, then **Project**.
2. In the **New Project** dialog that appears, under your desired language (VB.NET or C#), click the **Reporting** node.
3. Select the type of report application that you want to add (for information on the differences, see Report Types):
    - ActiveReports 14 Page Report Application
    - ActiveReports 14 RDL Report Application
    - ActiveReports 14 Section Report Application (xml-based)
    - ActiveReports 14 Section Report Application (code-based)



4. In the **Name** field, enter a name for the report application, and click **OK.** The selected report type is added to your project.
5. Go to the Visual Studio **Solution Explorer** and double-click Form1.cs or Form1.vb. Notice that the Viewer control already appears on the form.



Add Fields in Reports

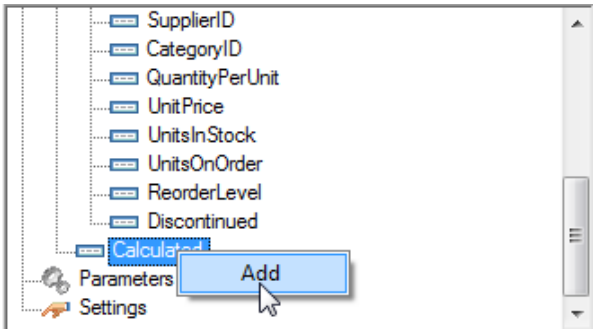
Fields provide data to display on a report page. ActiveReports has two types of fields; a bound or database field and a calculated field.

- **Bound or Database field:** A field where the value is returned by a query. See the **Query** dropdown in the Dataset Dialog for further information on queries in page reports and RDL reports.
- **Calculated field:** A field where the value is an expression created with functions, formulas and operators. Use the following instructions to add calculated fields in a report.

To create a calculated field in a Section Report

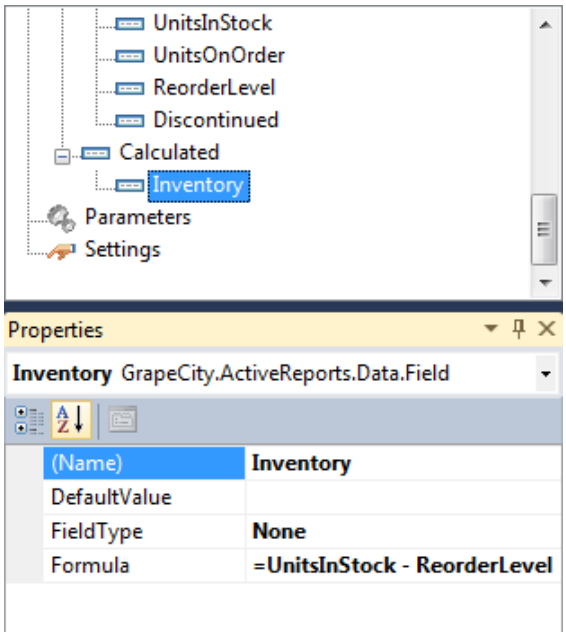
In a section report, once you connect to a data source, bound fields automatically appear under the Fields > Bound node in the Report Explorer. However, you have to add calculated fields manually under the Fields > Calculated node. The following steps guide you through the process.

1. In the Report Explorer, expand the Fields node.
2. Right-click the **Calculated** node and select Add. This action creates an unbound field with a default name like field1.



3. In the Report Explorer, with field1 selected, go to the Properties Window and set a value for the field in the **Formula ('Formula**

Property' in the on-line documentation) property. For e.g., for a calculated field Inventory, in the Formula field, enter the expression **=UnitsInStock - ReorderLevel**.



You can change the name of the field in the **Name ('Name Property' in the on-line documentation)** property.

4. Drag the field from the Calculated node onto the detail section of the report. This action creates a TextBox object, and sets its DataField property to the name of the calculated field.

Note: You can also add C# expressions in a Bound Field's DataField property to modify it. See Add Field Expressions for more information.

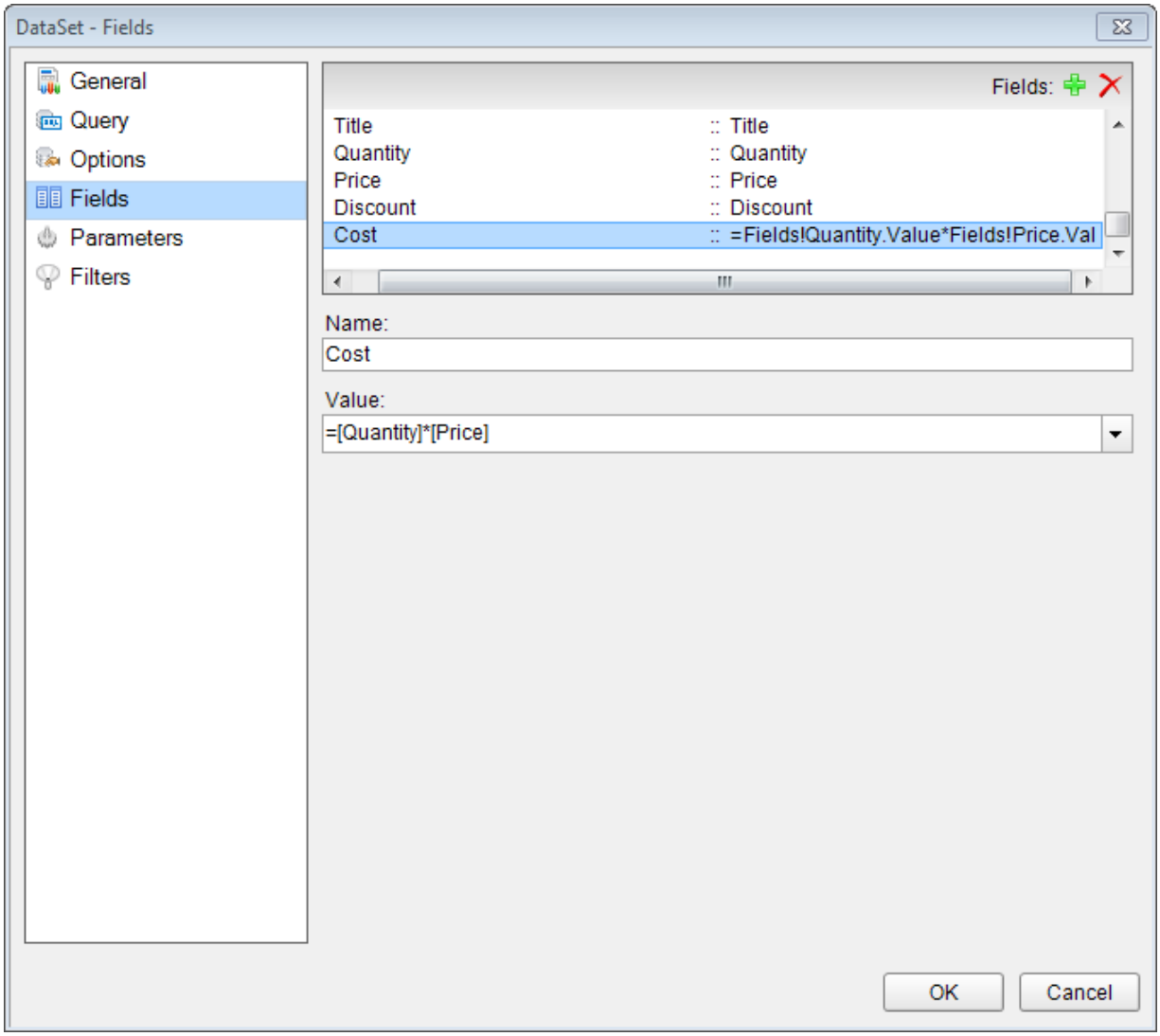
To create a calculated field in a Page Report/RDL Report

In a page report or a RDL report, all fields irrespective of their type appear under the corresponding DataSet node in the Report Explorer. To create a calculated field, you can add the new field in the DataSet dialog.

The following steps guide you through the process.

Note: These steps assume that you have added a DataSet in your report. See Add a Dataset for further information.

1. In the Report Explorer, right-click the data set node and select **Edit**.
2. In the DataSet dialog that appears, go to the **Fields** page and click the Add (+) button to add an empty field to the list.



3. Under **Name**, enter the name of the field. By default it appears as Field1.
4. Under **Value**, click the dropdown arrow and select <Expression...>, to open the Expression Editor dialog.
5. In the Expression Editor dialog, create an expression you want to use as the value for the calculated field. For e.g., for a calculated field Cost, in the Formula field, enter the expression **=[Quantity]*[Price]**. See Expressions for further information.
6. Click **OK** to close the Expression Editor and then the DataSet dialogs.
7. From the Report Explorer, drag the calculated field from that now appears as a field under the DataSet node onto the design surface. This action creates a TextBox object, and sets its Value property to the name of the calculated field expression.

Upgrade Reports

This section summarizes information on migration of ActiveReports, importing Microsoft Access reports, Crystal reports, and Microsoft Excel; and coexistence of ActiveReports .NET designers of different versions.

Topic	Content
Breaking Changes	Describes changes from the previous version
Migration Types	Describes the migration types and layouts

Migrating from Previous Versions	Describes migration from previous versions of ActiveReports for .NET
Migrating Execution Environment	Describes about migration of execution environment
Migrating from ActiveReports 2 COM	Describes migration from ActiveReports COM (the ActiveX version)
ActiveReports 2 COM versus ActiveReports for .NET	Describes the difference between ActiveReports COM and ActiveReports for .NET
Coexistence of ActiveReports Designers	Describes the compatibility of different versions of ActiveReports designers and Visual Studio

Breaking Changes

Breaking changes from ActiveReports 13 to ActiveReports 14

Installer

Simplified installation

- The GAC installation is removed. You no more need administrator previleges to install or update references.

Assemblies moved to NuGet - https://www.nuget.org/packages/

- NuGet packages are available instead of assemblies, so now you should install the latest NuGet packages available on the website. Installing a package automatically adds references to the necessary dlls.
- All dependencies are also available as NuGet packages through installer.

JSViewer and Web Designer are now available on NPM - https://www.npmjs.com/package/

Samples are moved to GitHub - https://github.com/activereports/

Removed HTML5Viewer

You should now use the improved JSViewer (available with Professional Edition).

Improved WebViewer

Changed UI of Html mode since we are using the new JSViewer internally.

ReportService settings in web.config (ActiveReportsXX section) are ignored.

Moved ReportsFolder property to the WebViewer control properties.

Changed WebViewer component API

- Removed method: ClearCachedReport
- Removed following events:
  - LocateDataSource**1**
  - LocateCredentials
- Removed following properties:
  - HtmlExportOptions
  - MaxReportRunTime
  - PdfExportOptions
  - SlidingExpirationIntervals

The WebViewer and JSViewer are supported only in the Integrated pipeline mode. You will get PlatformNotSupportedException on using these Viewers in Classic pipeline mode.

Dropped Oracle Data Provider

The Oracle data provider is no more available since System.Data.OracleClient is deprecated. If you want, you can still add this data provider to your application. The sample OracleDataProvider elaborates how to do that.

Modified Text Rendering

The text align is changed from Left to Center in last row of paragraph, if the last row contains only one character, and if the following properties are set: TextAlign = Justify and TextJustify=DistributeAllLines. This change can be observed in designer, preview, and PDF, Word, Excel, and Image exports.

Changed PdfRenderingExtension (RDL reports)

PDF export dialog

- Hid GDI-related properties: WatermarkFont > GdiCharSet and WatermarkFont > GdiVerticalFont.
- Removed OptimizeStatic property.

API

- Removed CertificateRawData property.

Changed SVG Rendering extension

Removed internal assembly GrapeCity.ActiveReports.Export.Svg. Now, **GrapeCity.ActiveReports.Core.Export.Svg.Page** assembly is used internally to export complex report items such as charts, map, etc. to HTML as SVG content.

Changed ImageRenderingExtension

- Dropped EMF output format (updated ImageType enum).
- Removed ColorDepth property from Settings class. This property was marked obsolete in ActiveReports 9.

Removed following assemblies

GrapeCity.ActiveReports.Core.Diagnostics.dll

RdfRenderingExtension

RdfRenderingExtension is now obsolete.

[1]Use **SetLocateDataSource ('SetLocateDataSource Method' in the on-line documentation)** method of the report service instead, as shown:

```
Application_Start(object sender, EventArgs e)        {
this.UseReporting(settings =>
    {
      settings.UseFileStore(new DirectoryInfo(Server.MapPath("~")));
      settings.UseCompression = true;
      settings.SetLocateDataSource(args => { return LoadData(); });
    });
}
```

Also, you can set the **LocateDataSource** event handler for a report directly as in the following code example.

```
pageReport.Document.LocateDataSource += new LocateDataSourceEventHandler(Document_LocateDataSource);
webViewer.Report = pageReport;
```

Breaking changes from ActiveReports 12 to ActiveReports 13

1. Following is the list of changes in public classes and namespaces location:

Class/Namespace	ActiveReports 12	ActiveReports 13	Impact
ResourceLocator, DefaultResourceLocator	GrapeCity.ActiveReports.Extensibility.v12	GrapeCity.ActiveReports.Core.Rdl	On creating own resource locator
GrapeCity.ActiveReports.PageReportModel.*	GrapeCity.ActiveReports.v12	GrapeCity.ActiveReports.Core.Rdl	On creating reports dynamically
GrapeCity.ActiveReports.Rendering.IO.*	GrapeCity.ActiveReports.v12	GrapeCity.ActiveReports.Core.Rendering	On rendering to any rendering extension
GrapeCity.ActiveReports.Extensibility.Rendering.Components.*	GrapeCity.ActiveReports.Extensibility.v12	GrapeCity.ActiveReports.Core.Rendering	On creating own rendering extensions or custom report items
GrapeCity.ActiveReports.Extensibility.Data.*	GrapeCity.ActiveReports.Extensibility.v12	-	On using own data providers
GrapeCity.ActiveReports.Dashboard.*	GrapeCity.ActiveReports.Dashboard.v12	GrapeCity.ActiveReports.Core.Rendering	-
GrapeCity.ActiveReports.Calendar.*	GrapeCity.ActiveReports.Calendar.v12	Calendar is moved to samples	-
GrapeCity.ActiveReports.Extensibility.Rendering.IRenderingExtension	GrapeCity.ActiveReports.Extensibility.v12	GrapeCity.ActiveReports	On creating own rendering extensions

GrapeCity.ActiveReports.ReportData.DataProviders.*	GrapeCity.ActiveReports.v12	GrapeCity.ActiveReports.Core.DataProviders	On using CSV, JSON, Object, Xml, DataSet data providers
GrapeCity.ActiveReports.ArsClient.*	GrapeCity.ActiveReports.ArsClient.v12	*removed*	-
GrapeCity.ActiveReports.OracleClient.*	GrapeCity.ActiveReports.OracleClient.v12	GrapeCity.ActiveReports.Core.DataProviders	-

2. The following assemblies have been removed:
   - GrapeCity.ActiveReports.ArsClient
   - GrapeCity.ActiveReports.Calendar
   - GrapeCity.ActiveReports.Dashboard
   - GrapeCity.ActiveReports.Export.Document
   - GrapeCity.ActiveReports.Export.Image.Unsafe
   - GrapeCity.ActiveReports.Export.Xaml
   - GrapeCity.ActiveReports.Extensibility
3. Following ActiveReports Core assemblies have been added:
   - GrapeCity.ActiveReports.Core.DataProviders
   - GrapeCity.ActiveReports.Core.Drawing.Gdi
   - GrapeCity.ActiveReports.Core.Rdl
   - GrapeCity.ActiveReports.Core.Rendering
   - GrapeCity.ActiveReports.Core.Export.Text.Page
   - GrapeCity.ActiveReports.Core.Export.Excel.Page
   - GrapeCity.ActiveReports.Core.Export.Html.Page
4. Assembly GrapeCity.ActiveReports.Diagnostics is renamed to **GrapeCity.ActiveReports.Core.Diagnostics**.
5. Following assemblies are added that have back-ends for JsViewer and Web Designer components:
   - GrapeCity.ActiveReports.Aspnetcore.Viewer.dll
   - GrapeCity.ActiveReports.Aspnet.Viewer.dll
   - GrapeCity.ActiveReports.Aspnetcore.Designer.dl
   - GrapeCity.ActiveReports.Aspnet.Designer.dll
6. **Flash** and **Silverlight viewers** are now obsolete.
7. **HTML5 Viewer** is deprecated and will be obsolete in the future version.
8. **Custom Data Providers** have been modified as:
   - Removed extra classes and interfaces - DbCommand, DbConnection and others have been removed to reuse standard System.Data.* classes. Now implement System.Data.DataProviderFactory, System.Data.DbCommand, System.Data.DbConnection, System.Data.DbDataReader OR download any third-party implementation of those standard interfaces.
   - Simplified using custom data providers - The customer may just configure ActiveReports to start using of any ADO component.

     Note: Multivalue parameters, Credentials, and UI editor are available only if customer implements special adapter.
     To support setting of credentials and multivalue parameters, implement GrapeCity.ActiveReports.ReportData.DataProviders.DbConnectionAdapter.
   - Created data provider should be configured using ActiveReports.config - Specify name and data provider factory type. You may specify custom editor, adapter type (AdapterType attribute) and schema provider type (SchemaProviderType attribute).
9. **LocateDataSourceEventArgs** class which obtains data for LocateDataSource event has undergone following changes.
   - Following properties have been removed:
     - DataSetName:string
     - DataSourceName:string
     - Report:PageDocument
   - Following properties have been added:
     - DataSet:IDataSet
     - Report:ReportObjectModel.Report
     - Parameters:IReadonlyList<DataParameter>

ActiveReports 12 approach	ActiveReports 13 approach
`class LocateDataSourceEventArgs{`	`class LocateDataSourceEventArgs{`
`public object Data {get;set;}`	`public object Data {get;set;}`
`public string DataSetName {get;}`	**`public IDataSet DataSet {get;}`**
`public string DataSourceName {get;}`	**`public Report Report {get;}`**
`public PageDocument Report {get;}`	**`public IReadonlyList<DataParameter> Parameters {get;}`**
`}`	`}`

   Note: The ActiveReports 13 parameters are Query parameters, so if you want to obtain Subreport parameters (available in PageDocument in ActiveReports 12), you should map them to Query parameters first.
10. **Custom Report Items** have been modified as**:**
    - Most of the extra methods of ICustomReportItem interface (required for Flash) were dropped.
    - Now CRIs should implement IReportItemRenderersFactory interface.
11. **HTML RE ILinkProvider**

Cleaned ILinkProvider interface with few new properties.

12. **HTML RE**
    HTML RE no more inherits IDisposable interface with Dispose method from HtmlRenderingExtension class (for WebViewer).

Breaking changes from ActiveReports 11 to ActiveReports 12

- The GrapeCity.ActiveReports.Expressions namespace is now GrapeCity.Enterprise.Data.DataEngine.Expressions.
- The hybrid expressions like `=Theme.Constants("Name") & " - " & [OrderDate].Year` are now evaluated as 'render time' expressions.
- ActiveReports 12 is not supported in Microsoft Visual Studio 2010.
- Flash and Silverlight viewers are deprecated.

Breaking changes from Previous ActiveReports versions to ActiveReports 12

When you upgrade reports from previous versions of ActiveReports or Data Dynamics Reports, there are several breaking changes. In ActiveReports 12, data engine has been revamped to improve the data manipulation tasks such as sorting, filtering, and grouping.

Control Changes

The Excel Transformation Device option, the **File** menu item **Microsoft Excel WorkSheet - Data (XLS)**, is no longer available for RDL reports in the default export dialogs of the viewer and designer applications shipped with the product. For backward compatibility, the Excel Transformation Device API is still available, but it does not support the new Tablix control. In order to successfully export reports using the new Tablix control, please use the Excel Rendering Extension option, the **File** menu item **Microsoft Excel WorkSheet - Layout (XLS, XLSX).**

The Matrix data region has been replaced in the toolbox with the new Tablix data region. However, it is still available in the API for backward compatibility.

The **OleObject** control is now hidden by default in the toolbox for Section reports. To show this control in Visual Studio, open the GrapeCity.ActiveReports.config file and change the **EnableOleObject** value to **true**, and include this file with your application. You can find this file here: `C:\Program Files (x86)\Common Files\GrapeCity\ActiveReports 12`.

To show the OleObject control in the Designer control in your own end users designer applications, select the Designer control and, in the Properties window, change the **EnableOleObject** property to **True**.

The **WebViewer** control is now AJAX-based, and requires **ActiveReports.ReportService.asmx** to be in the root of the Web site or Web application. This is added automatically when you drop a WebViewer control on a Web form, or you can add it from the **Add New Item** dialog by selecting **ActiveReports 12 Web Service**, or manually by copying it from C:\Program Files (x86)\Common Files\GrapeCity\ActiveReports 12. **ExceptionOccurring**, **QueuingReport**, **ReportCreating** and **ReportDisposing** events are no longer available in **WebViewer** class.

The **Viewer** control no longer has Annotations turned on by default. To enable Annotations, set the **AnnotationDropDownVisible** property of the Viewer control to **True**. **DataDynamics.ActiveReports.Viewer.ReportViewer.MultiplePageMode** property is now integrated into **ViewType** property of **Viewer** class. **Viewer.ReportViewer.PaperColor** property has been removed. **Viewer.PageOffset** property's type has been changed from **Integer** to **System.Drawing.Point**.

The **Toolbar** is now a Windows ToolStrip. Please see the MSDN ToolStrip Class for more information.

In RDL and Page reports, the default behavior of the **Series Border Style** setting for Area, Pie and Doughnut chart types has been changed. Now, the default value of this setting is **None**, thus no style formatting is applied.

Other Changes

Rendering Extensions (Image): ColorDepth is an obsolete property of GrapeCity.ActiveReports.Export.Image.Page.Settings class.

Expressions: In Page report and RDL report expressions, "True" and "False" values are now handled as String, and not as Boolean values.
For example, `=IIF(Fields!FieldName.Value = "True", 1, 0)` is now an invalid expression when FieldName.Value returns a Boolean value, instead, use `=IIF(Fields!FieldName.Value = True, 1, 0)` expression.

Classes in Different Namespaces

In ActiveReports 12, some classes have been moved to different namespaces from previous versions of ActiveReports and Data Dynamics Reports. Drop down the table below to see some of the most commonly used classes that are in new namespaces.

Namespace Changes and Restructuring

Some of the changes that are not picked up by the upgrade tool may cause some issues in your code. The two most frequently encountered changes are:

- DataDynamics.ActiveReports.**ActiveReport** is now GrapeCity.ActiveReports.**SectionReport**
- DataDynamics.ActiveReports.Document.**Document** is now GrapeCity.ActiveReports.Document.**SectionDocument**

These are all of the assemblies and namespaces that have changed, with any major changes noted.

ActiveReports is now GrapeCity.ActiveReports.v12

- **ActiveReport** class is now called **SectionReport**.
- **BarWidth** property is now called **NarrowBarWidth**.

ActiveReports Namespace (previous versions)	ActiveReports 12 Namespace
DataDynamics.ActiveReports	• GrapeCity.ActiveReports • GrapeCity.ActiveReports.SectionReportModel • GrapeCity.ActiveReports.Data
DataDynamics.ActiveReports.DataSources	GrapeCity.ActiveReports.Data

DataDynamics.ActiveReports.Interop	GrapeCity.ActiveReports
DataDynamics.ActiveReports.Options	GrapeCity.ActiveReports.SectionReportModel

ActiveReports.Chart is now GrapeCity.ActiveReports.Chart.v12

ActiveReports Namespace (previous versions)	ActiveReports 12 Namespace
DataDynamics.ActiveReports.Chart	GrapeCity.ActiveReports.Chart
DataDynamics.ActiveReports.Chart.Annotations	GrapeCity.ActiveReports.Chart.Annotations
DataDynamics.ActiveReports.Chart.Graphics	GrapeCity.ActiveReports.Chart.Graphics

ActiveReports.Design is now GrapeCity.ActiveReports.Design.Win.v12

The **Report** property is now an Object that gets or sets a **GrapeCity.ActiveReports.Document.SectionDocument** or **GrapeCity.ActiveReports.Document.PageDocument**.

The **ColorTheme** property of the Designer class is deprecated.

ActiveReports Namespace (previous versions)	ActiveReports 12 Namespace
DataDynamics.ActiveReports.Design	GrapeCity.ActiveReports.Design
DataDynamics.ActiveReports.Design.ReportExplorer	GrapeCity.ActiveReports.ReportExplorer
DataDynamics.ActiveReports.Design.Toolbox	GrapeCity.ActiveReports.Design.Toolbox

ActiveReports.Document is now GrapeCity.ActiveReports.Document.v12

The **Document** class is now called **SectionDocument**.

ActiveReports Namespace (previous versions)	ActiveReports 12 Namespace
DataDynamics.ActiveReports	GrapeCity.ActiveReports
DataDynamics.ActiveReports.Document	• GrapeCity.ActiveReports.Document • GrapeCity.ActiveReports.Document.Section • GrapeCity.ActiveReports.Extensibility.Printing(GrapeCity.ActiveReports.Extensibility.v12)
DataDynamics.ActiveReports.Export	GrapeCity.ActiveReports.Export
DataDynamics.ActiveReports.Export.Html	GrapeCity.ActiveReports.Export.Html
DataDynamics.ActiveReports.Document.Annotations	GrapeCity.ActiveReports.Document.Section.Annotations

ActiveReports.HtmlExport is now GrapeCity.ActiveReports.Export.Html.v12

ActiveReports Namespace (previous versions)	ActiveReports 12 Namespace
DataDynamics.ActiveReports.Export.Html	GrapeCity.ActiveReports.Export.Html.Section

ActiveReports.PdfExport is now GrapeCity.ActiveReports.Export.Pdf.v12

ActiveReports Namespace (previous versions)	ActiveReports 12 Namespace
DataDynamics.ActiveReports.Export.Pdf	GrapeCity.ActiveReports.Export.Pdf.Section
DataDynamics.ActiveReports.Export.Pdf.Signing	GrapeCity.ActiveReports.Export.Pdf.Section.Signing

ActiveReports.RtfExport is now GrapeCity.ActiveReports.Export.Word.v12

ActiveReports Namespace (previous versions)	ActiveReports 12 Namespace
DataDynamics.ActiveReports.Export.Rtf	GrapeCity.ActiveReports.Export.Word.Section

ActiveReports.Silverlight is now GrapeCity.ActiveReports.Viewer.Silverlight.v12

ActiveReports Namespace (previous versions)	ActiveReports 12 Namespace
DataDynamics.ActiveReports	GrapeCity.ActiveReports

ActiveReports.TextExport is now GrapeCity.ActiveReports.Export.Xml.v12

ActiveReports Namespace (previous versions)	ActiveReports 12 Namespace
DataDynamics.ActiveReports.Export.Text	GrapeCity.ActiveReports.Export.Xml.Section

ActiveReports.TiffExport is now GrapeCity.ActiveReports.Export.Image.v12

ActiveReports Namespace (previous versions)	ActiveReports 12 Namespace

DataDynamics.ActiveReportsExport.Tiff	GrapeCity.ActiveReports.Export.Image.Tiff.Section

ActiveReports.Viewer is now GrapeCity.ActiveReports.Viewer.Win.v12

- The **History** class is now an interface, **IHistoryApi**, that resides in the **GrapeCity.Viewer.Common** namespace.
- The **SearchResultsForeColor** property now gets applied as the border around the searched text.
- The **DisplayUnits** and **RulerVisible** properties of the Viewer class have been removed as the Viewer no longer uses a ruler.
- The **TabTitleLength** property of the Viewer class is not available as the tab function of the Viewer has been removed.
- The **ViewerToolbar.DisplayToolTips** property of the Viewer.ViewerToolbar class is now **ViewerToolbar.ToolStrip.ShowItemToolTips**.
- The **ViewerToolbar.Enabled** property of the Viewer.ViewerToolbar class is now **ViewerToolbar.ToolStrip.Enabled**.
- The **ViewerToolbar.Visible** property of the Viewer.ViewerToolbar class is now **ViewerToolbar.ToolStrip.Visible**.
- The **TargetView** enumeration now has two enumeration values (Primary and Secondary).
- The **ToggleVisibility()** method is now **Visible** property that determines whether sidebar is visible or hidden.
- The **Print ('Print Method' in the on-line documentation)** method is implemented as an extension method of the **PrintExtension.Print ('Print Method' in the on-line documentation)** method, which is present in Grapecity.ActiveReport namespace of GrapeCity.ActiveReports.Viewer.Win.v12 assembly.

ActiveReports Namespace (previous versions)	ActiveReports 12 Namespace
DataDynamics.ActiveReports.Toolbar	The viewer now uses Visual Studio ToolStrips. Please see MSDN ToolStrip Class for more information.
DataDynamics.ActiveReports.Viewer	• GrapeCity.ActiveReports.Viewer.Win • GrapeCity.Viewer.Common

Note: GrapeCity.ActiveReports.Viewer.Win.v12.dll does not get added automatically to the project references when the report layout is added. You need to either add the Viewer control or manually add the reference to this assembly.

ActiveReports.Web is now GrapeCity.ActiveReports.Web.v12

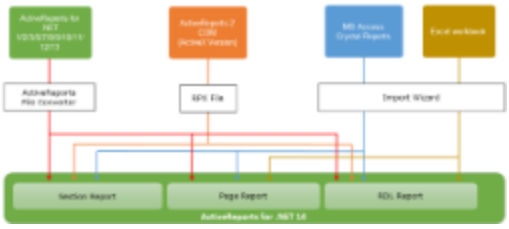
The **Report** property is now an Object that gets or sets a **SectionDocument** or **ReportDocument**.

ActiveReports Namespace (previous versions)	ActiveReports 12 Namespace
DataDynamics.ActiveReports.Web	GrapeCity.ActiveReports.Web
DataDynamics.ActiveReports.Web.Controls	GrapeCity.ActiveReports.Web.Controls
DataDynamics.ActiveReports.Web.ExportOptions	GrapeCity.ActiveReports.Web.ExportOptions
DataDynamics.ActiveReports.Web.Handlers	GrapeCity.ActiveReports.Web.Handlers

ActiveReports.XlsExport is now GrapeCity.ActiveReports.Export.Excel.v12

ActiveReports Namespace (previous versions)	ActiveReports 12 Namespace
DataDynamics.ActiveReports.Export.Xls	GrapeCity.ActiveReports.Export.Excel.Section
DataDynamics.SpreadBuilder	GrapeCity.SpreadBuilder
DataDynamics.SpreadBuilder.Cells	GrapeCity.SpreadBuilder.Cells
DataDynamics.SpreadBuilder.Imaging	GrapeCity.SpreadBuilder.Imaging
DataDynamics.SpreadBuilder.Printing	GrapeCity.SpreadBuilder.Printing
DataDynamics.SpreadBuilder.Style	GrapeCity.SpreadBuilder.Style

Migration Types



Migration from ActiveReports for .NET (1 / 2 / 3 / 6 / 7 / 8/ 9/ 10/11/12/13)

You can migrate to all the three types of reports - Section report, Page, or an RDL report in ActiveReports 14 using the file converter.

Migrating from ActiveReports 2 COM

You can migrate only the design information of ActiveReports 2 COM by saving the report as an RPX file. You can load and use the file after migrating to ActiveReports for .NET 11. No special migration tools are required.

Migrating from MS Access, Crystal Reports, and Excel workbook

You can use the Import Wizard to migrate reports to RPX files (Section reports) or RDLX files (Page and RDL reports). When migrating from MS Excel, it is possible to migrate only to RDLX file. You can load and use the file after migrating to ActiveReports for .NET 12.

Migrate from Previous Versions

This section explains how to migrate projects created with previous versions of ActiveReports to ActiveReports 14.

This section contains information about:

Topic	Content
ActiveReports Version Up History	Describes the history of changes in the ActiveReports for .NET product.
Migrate to ActiveReports 14	Describes how to migrate ActiveReports versions starting from ActiveReports 1 to ActiveReports 14.
Reference Migration	Describes how to migrate assembly (DLL) references.
License Migration	Describes how to migrate license information in a project.
ds Variable	Describes how to migrate the public variable 'ds' automatically generated by the previous version of ActiveReports.
WebViewer Migration	Describes how to migrate the WebViewer control included in ActiveReports Professional Edition.
ActiveX Viewer Migration	Describes how to migrate the ActiveX Viewer included in ActiveReports 1, 2, and 3.
Compatibility Guidelines	Describes compatibility issues between ActiveReports 14 and older versions of ActiveReports.

ActiveReports Version Up History

Please refer to this link for the history of changes that ActiveReports for .NET have undergone due to version upgrade.

ActiveReports File Converter

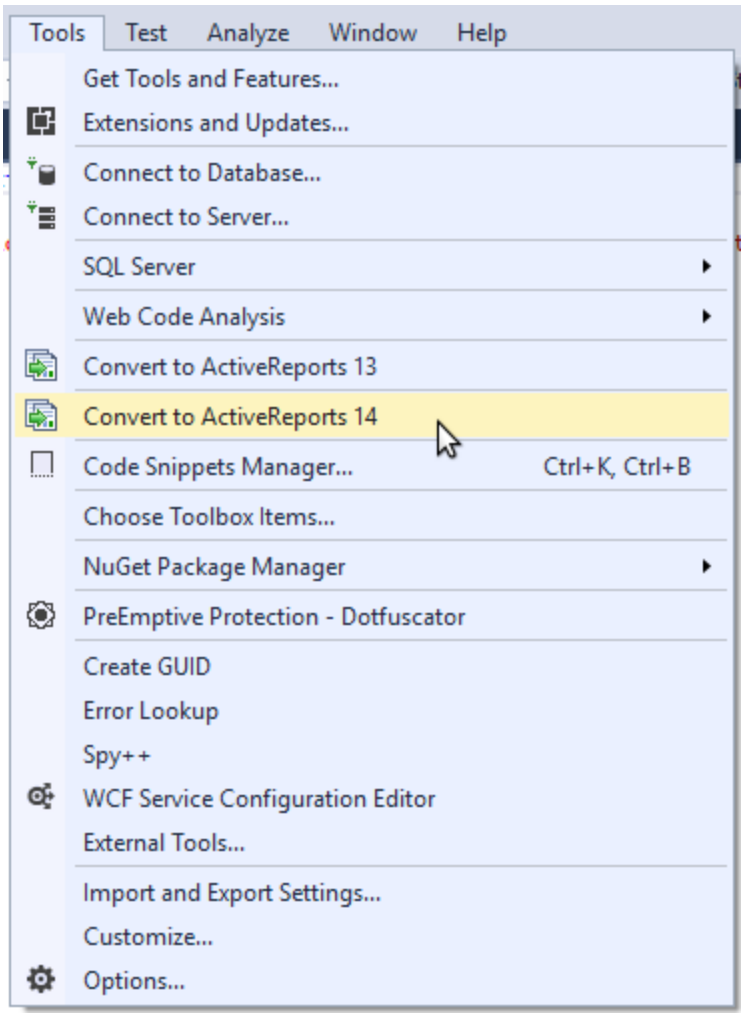
The ActiveReports file converter allows you to upgrade your existing reports from previous versions of ActiveReports (ActiveReports 13, 12, 11, 10, 9, 8, 7, 6, 3, 2, and 1) to the latest version.

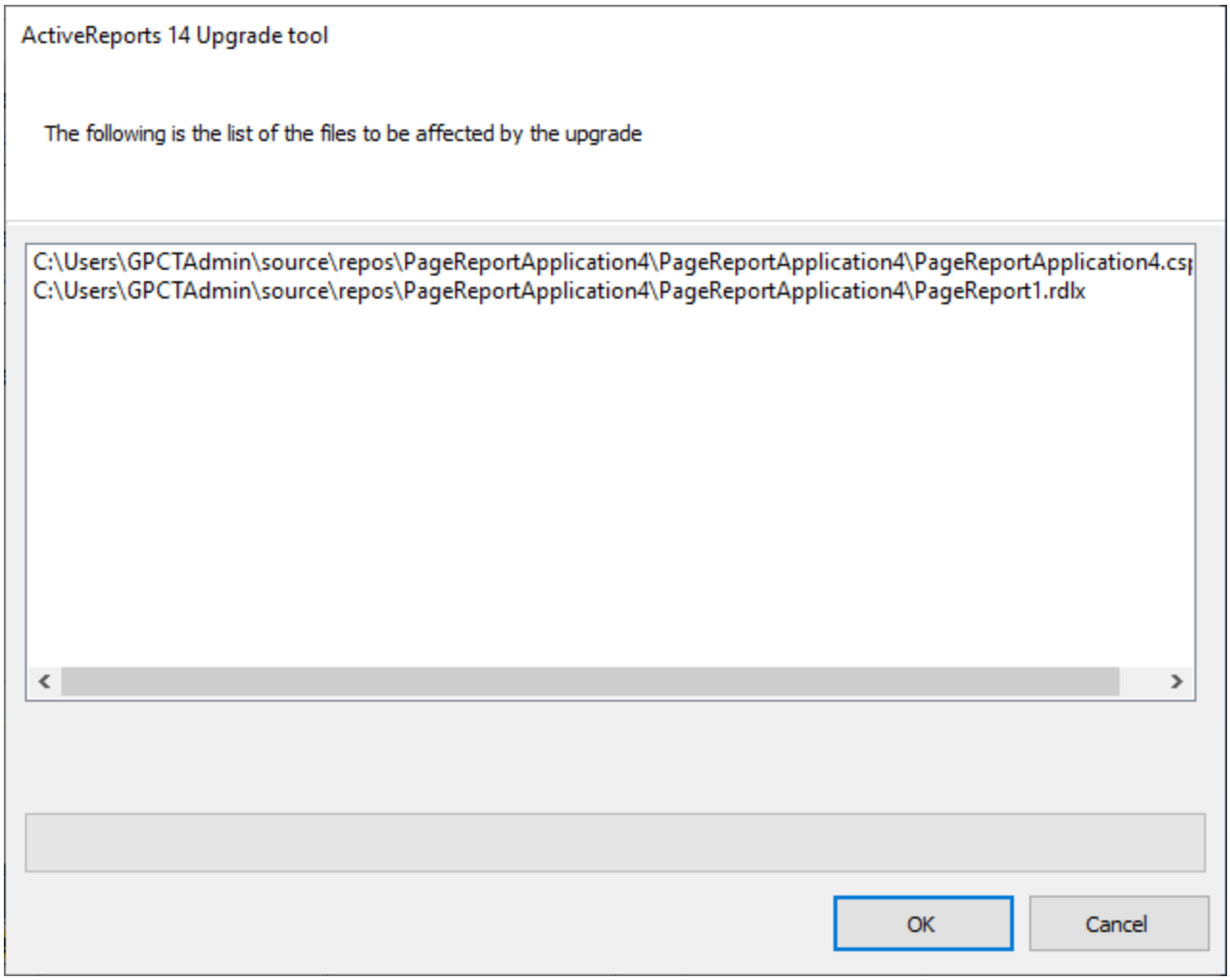
To use the file converter

1. In Visual Studio, open an existing ActiveReports project.

   > ⚠ **Caution:** If you are migrating a project from a different version of Visual Studio, a Visual Studio conversion wizard will run automatically to convert the project.

2. From the Visual Studio **Tools** menu, select **Convert to ActiveReports 14**. In the ActiveReports 14 Upgrade tool window that appears, you can see a list of report files to be converted.





File Type	Extension
Report file	*.rdlx, *.rpx, *.vb, *.cs
Form with a viewer	*.vb, *.cs, *.xaml
ASP.NET Web Form with a WebViewer	*.aspx, *.aspx.vb, *.aspx.cs
Project file	*.vbproj, *.csproj, Web.confg
License file	licenses.licx

3. Click **OK** to convert the files. After the files are successfully converted, all of the reference files are updated and the reports are converted to C# or Visual Basic class files. The old files are copied to **ARConverterBackup** folder generated under the root folder of the project. You can delete these files from the project if you do not need them.

> ⚠ **Caution:** When converting from version 6 or below, you may observe the following warnings on migrating a form with the viewer control. These warnings are due to API modifications in ActiveReports 7 or above. You can ignore these warnings or delete them manually.
>
> - 'Public Property Text() As String' is an old format: 'Not used anymore'
> - 'Public Property TabTitleLength() As Integer' is an old format: 'Not used anymore'

> **Note:** In addition to the above conversions, you may also observe some formatting changes, such as code breaks, while using this converter tool.

4. After running the converter, you need to manually update all the project references. For more details, see Migrating from Previous Versions.

Migrate to ActiveReports 14

The following steps explain how to migrate previous ActiveReports versions to ActiveReports 14.

1. From the Visual Studio **Tools** menu, select **Convert to ActiveReports 14**.
   - If the assembly version of reference files does not automatically change to ActiveReport 13 after running the converter, you need to manually update the reference files. For more details, see Reference Migration.
   - If the license information is not displayed correctly after running the converter, you need to manually update the license information. For more details, see License Migration.
2. See details below to migrate any of the following versions that contain specific classes or properties:

ActiveReports 1

- If you export to PDF and use the **Version** property, you need to change the casing in the PdfVersion enumerated values as follows:

  Before migration: PDFVersion.PDF13
  After migration: PdfVersion.Pdf13

- If you are using the public variable 'ds' within the report, see ds Variable.
- If you are using the WebViewer control, see WebViewer Migration.
- If you are using the ActiveX viewer (ARVIEW2.CAB file) in a Web application, see ActiveX Viewer Migration.

ActiveReports 2

- If you are using the public variable 'ds' within the report, see ds Variable.
- If you are using the WebViewer control, see WebViewer Migration.
- If you are using the ActiveX viewer (ARVIEW2.CAB file) in a Web application, see ActiveX Viewer Migration.

ActiveReports 3

- If you are using the ActiveReport3 class in your project, you need to change **ActiveReport3** class to **SectionReport** class.

  Before migration: DataDynamics.ActiveReports.ActiveReport3
  After migration: GrapeCity.ActiveReports.SectionReport

- If you are migrating from an initial version of ActiveReports 3 and use the Barcode control's Style and BackColor properties, then the size of the Barcode control may change on migration. You can set the BackColor property of the Barcode control to White, or manually change the size.

Property	Value
Style ('Style Property' in the on-line documentation)	QRCode, Code49, JapanesePostal, Pdf417, EAN128FNC1 (Any of these)
BackColor ('BackColor Property' in the on-line documentation)	System.Drawing.Color.Transparent

- If you are using the public variable 'ds' within the report, see ds Variable.
- If you are using the WebViewer control, see WebViewer Migration.

- If you are using the ActiveX viewer (ARVIEW2.CAB file) in a Web application, see ActiveX Viewer Migration.

From ActiveReports 6 to ActiveReports 12

If you are using the WebViewer control, see WebViewer Migration.

3. Check the Compatibility Guidelines.
4. From the **Build** menu, select **Rebuild Solution** to rebuild the entire solution.

> **Note:** Between versions 1 and 14, we have updated several class names. This may cause syntax errors on migration. To resolve these errors, see the Breaking Changes topic.

Reference Migration

This topic explains how to upgrade your project references (.dlls) in your project. ActiveReports 14 does not support references to previous versions, so you need to remove the old references and add new ones.

To upgrade your references

1. In Visual Studio, open an existing ActiveReports project.
2. In the Project Explorer, expand the References node, and remove the old references.
   Here are the ActiveReports version numbers by product name.

Product Name	Assembly Version
ActiveReports 1	3.x.x.xxxx
ActiveReports 2	4.x.x.xxxx
ActiveReports 3	5.x.x.xxxx
ActiveReports 6	6.x.x.xxxx
ActiveReports 7	7.x.x.xxxx
ActiveReports 8	8.x.x.xxxx
ActiveReports 9	9.x.x.xxxx
ActiveReports 10	10.x.x.xxxx
ActiveReports 11	11.x.x.xxxx
ActiveReports 12	12.x.x.xxxx
ActiveReports 13	13.x.x.xxxx
ActiveReports 14	14.x.x.xxxx

3. Add references to ActiveReports 14. In the Solution Explorer, right-click the References node and select **Manage NuGet Packages**.
4. In the Package source on top right, select **nuget.org**.
5. Click Browse tab on top left and search for the packages corresponding to the removed assemblies in Step 2.
6. On the right panel, click **Install**.

7. In the License Acceptance dialog, select I Accept to proceed the installation.

   Please see Installed Files for details on the list of assemblies required in ActiveReports 14.

License Migration

This topic explains how to migrate license information for your ActiveReports project from an older version.

The process of licensing an ActiveReports application is different for each version. Migrate license information for Windows or Web projects as follows.

Windows Application

In your Windows application, check that the licenses.licx file contains the appropriate licensing strings. When upgrading from a previous version, remove the old information and update the license strings in the licenses.licx file. You can find the full list of license strings that you may need for licensing in the License Your ActiveReports topic **Required references in the licenses.licx file (for Standard and Professional Editions)** section.
Here are how the licensing strings appear before and after migration.

- Before migration (ActiveReports 1/2/3)
  ```
  DataDynamics.ActiveReports.ActiveReport, ActiveReports
  ```
- Before migration (ActiveReports 6)
  ```
  DataDynamics.ActiveReports.ActiveReport, ActiveReports
  DataDynamics.ActiveReports.Viewer.Viewer, ActiveReports.Viewer6
  DataDynamics.ActiveReports.Export.Pdf.PdfExport, ActiveReports.PdfExport
  ```
- Before migration (ActiveReports 7)
  ```
  GrapeCity.ActiveReports.SectionReport, GrapeCity.ActiveReports.
  GrapeCity.ActiveReports.Viewer.Win.Viewer, GrapeCity.ActiveReports.Viewer.Win.
  GrapeCity.ActiveReports.Export.Pdf.Section.PdfExport,
  GrapeCity.ActiveReports.Export.Pdf.
  ```
- Before migration (ActiveReports 9)
  ```
  GrapeCity.ActiveReports.SectionReport, GrapeCity.ActiveReports.v9
  GrapeCity.ActiveReports.Viewer.Win.Viewer, GrapeCity.ActiveReports.Viewer.Win.v9
  GrapeCity.ActiveReports.Export.Pdf.Section.PdfExport,
  GrapeCity.ActiveReports.Export.Pdf.v9
  ```
- Before migration (ActiveReports 10)
  ```
  GrapeCity.ActiveReports.SectionReport, GrapeCity.ActiveReports.v10
  GrapeCity.ActiveReports.Viewer.Win.Viewer, GrapeCity.ActiveReports.Viewer.Win.v10
  GrapeCity.ActiveReports.Export.Pdf.Section.PdfExport,
  GrapeCity.ActiveReports.Export.Pdf.v10
  ```
- Before migration (ActiveReports 11)
  ```
  GrapeCity.ActiveReports.SectionReport, GrapeCity.ActiveReports.v11
  GrapeCity.ActiveReports.Viewer.Win.Viewer, GrapeCity.ActiveReports.Viewer.Win.v11
  GrapeCity.ActiveReports.Export.Pdf.Section.PdfExport,
  GrapeCity.ActiveReports.Export.Pdf.v11
  ```
- Before migration (ActiveReports 12)
  ```
  GrapeCity.ActiveReports.SectionReport, GrapeCity.ActiveReports.v12
  GrapeCity.ActiveReports.Viewer.Win.Viewer, GrapeCity.ActiveReports.Viewer.Win.v12
  ```

```
GrapeCity.ActiveReports.Export.Pdf.Section.PdfExport,
GrapeCity.ActiveReports.Export.Pdf.v12
```

- Before migration (ActiveReports 13)

```
GrapeCity.ActiveReports.SectionReport, GrapeCity.ActiveReports
GrapeCity.ActiveReports.Viewer.Win.Viewer, GrapeCity.ActiveReports.Viewer.Win
GrapeCity.ActiveReports.Export.Pdf.Section.PdfExport,
GrapeCity.ActiveReports.Export.Pdf
```

- After migration (ActiveReports 14)

```
GrapeCity.ActiveReports.SectionReport, GrapeCity.ActiveReports
GrapeCity.ActiveReports.Viewer.Win.Viewer, GrapeCity.ActiveReports.Viewer.Win
GrapeCity.ActiveReports.Export.Pdf.Section.PdfExport,
GrapeCity.ActiveReports.Export.Pdf
```

Web Application

When migrating from ActiveReports versions 1, 2, and 3, you can find the license strings in the Web.config file. In ActiveReports 6 or above, the license strings are in the licenses.licx file. You can remove the old information from the Web.config file and update the necessary licensing strings in the project. For more details, please see the License Your ActiveReports topic **Required references in the licenses.licx file (for Standard and Professional Editions)** section.

- Before migration (Web.Config file)

  The following information is not required in ActiveReports 6 or above.

```
<Configuration>
      (...)
<appSettings>
  <add key="DataDynamicsARLic" value="xxxxxxx,xxxxxxxxxx,xxxxxxxxxx,xxxxxxxxxxxxx"
/>
</appSettings>
     (...)
</Configuration>
```

- After migration (licenses.licx file)

  For example, if you are creating a Web application that uses the WebViewer, add the following strings to the licenses.licx file in the project.

```
GrapeCity.ActiveReports.SectionReport, GrapeCity.ActiveReports
GrapeCity.ActiveReports.Export.Pdf.Section.PdfExport,
GrapeCity.ActiveReports.Export.Pdf
GrapeCity.ActiveReports.Web.WebViewer, GrapeCity.ActiveReports.Web
```

- You may also face an error due to the compiled license information of previous version mentioned in the Bin folder of the project. If the error occurs even after including the correct information in the license file, remove App_Licenses.dll available in the Bin folder

ds Variable

Migration Requirements

In ActiveReports 2 or below, when you create a data connection using the **Report Data Source** dialog, a public variable 'ds' is generated automatically by the report designer. This variable is not generated in ActiveReports 3 or above. If you use this variable in your code for making data connection settings of subreport, you need to change the code as follows.

To migrate code with public variable 'ds'

Before migration

```
Me.ds.ConnectionString 'Visual Basic

this.ds.ConnectionString; //C#
```

After migration

```
CType(Me.DataSource, GrapeCity.ActiveReports.Data.OleDBDataSource).ConnectionString
'Visual Basic

((GrapeCity.ActiveReports.Data.OleDBDataSource)(this.DataSource)).ConnectionString; //C#
```

WebViewer Migration

This topic explains how to upgrade the WebViewer control available in the Professional edition of ActiveReports.

The ASP.NET Web form or the Web.config file included in the WebViewer control is upgraded automatically on running the ActiveReports file converter. However, the converter does not upgrade the Flash viewer since flash viewer is obsolete.

The license file should also get updated while conversion, but in some cases due to the project configuration, you may need to manually update the license file. For more information on upgrading license, see License Migration.

ActiveX Viewer Migration

This topic explains how to migrate the ActiveX viewer included in ActiveReports 3, 2, and 1.

Migration Requirements

In ActiveReports 6 or above, ActiveX viewer (ARVIEW2.CAB) is not provided with the product installer. You need to migrate a web application that includes ActiveX viewer on the Web form or contains the WebViewer with ViewerType property set to ActiveXViewer. In ActiveReports 14, you can use the PDF export feature instead of ActiveX Viewer.

> **Note: To use the ActiveX viewer associated with previous version**
> In ActiveReports 14, you cannot use the ActiveX viewer of previous version. You can obtain the expected display or print results as per the report, however, such usage methods are not included in the product operational guarantee.

Alternative Approach

For both Professional and Standard editions, use PDF export to preview or print a report with quality similar to that observed with ActiveX viewer.

- For ActiveReports Professional edition
  - WebViewer (PDF)
  - PDF Export
- For ActiveReports Standard edition
  - PDF Export

Precautions

You can use print and preview features with same quality as mentioned in alternate approach, however, some print features are disabled.

	Not available in ActiveReports 14		Available in ActiveReports 14	
Important Features	ActiveX Viewer	WebViewer (ActiveX) Professional Edition	WebViewer (PDF) Professional Edition	PDF Export
Direct printing from client printer	○	○	○	○
Direct printing from client printer without preview	○	○	○	○
Direct printing from client printer without preview (Windows print dialog is hidden ※1)	○	○	×	×
Direct printing from client printer after changing settings (printer type, paper size etc.)	○ (※3)	○ (※3)	○ (※2)	○ (※2)
Auto scaling	×	×	×	×
Automatic setting of page orientation	×	×	×	×

※1: In client printing, if you click Print button, Windows Print dialog is displayed. In ActiveX viewer you can hide the Windows Print dialog.

※2: In Windows Print dialog, similar to general Office application, you can select paper size or orientation.

※3: In addition to point ※2, the developer can set the desired page size or page orientation in the printer through client scripting before the user displays the Windows print dialog.

To migrate project that uses ActiveXViewer as the ViewerType

If the ViewerType property is set to ActiveXViewer, you need to change it to AcrobatReader. For example, modify the code marked in red like the following code.

Before migration

Visual Basic

```
' Set the type of the selected view in ViewerType property of the WebViewer.
Dim selection As String = Me.cboViewerType.Items(Me.cboViewerType.SelectedIndex).Text
Select Case selection
 Case "AcrobatReader"
  Me.arvWebMain.ViewerType = DataDynamics.ActiveReports.Web.ViewerType.AcrobatReader
 Case "ActiveX viewer"
  Me.arvWebMain.ViewerType = DataDynamics.ActiveReports.Web.ViewerType.ActiveXViewer
 Case "HTML viewer"
  Me.arvWebMain.ViewerType = DataDynamics.ActiveReports.Web.ViewerType.HtmlViewer
 Case "RawHtml"
  Me.arvWebMain.ViewerType = DataDynamics.ActiveReports.Web.ViewerType.RawHtml
End Select
```

C#

```
// Set the type of the selected view in ViewerType property of the WebViewer.
{
    string selection = this.cboViewerType.Items(this.cboViewerType.SelectedIndex).Text;
    switch (selection) {
        case "AcrobatReader":
            this.arvWebMain.ViewerType =
DataDynamics.ActiveReports.Web.ViewerType.AcrobatReader;
            break;
        case "ActiveX viewer":
            this.arvWebMain.ViewerType =
DataDynamics.ActiveReports.Web.ViewerType.ActiveXViewer;
            break;
        case "HTML viewer":
            this.arvWebMain.ViewerType =
DataDynamics.ActiveReports.Web.ViewerType.HtmlViewer;
            break;
        case "RawHtml":
            this.arvWebMain.ViewerType =
DataDynamics.ActiveReports.Web.ViewerType.RawHtml;
            break;
    }
}
```

After migration

Visual Basic

```
' Set the type of the selected view in ViewerType property of the WebViewer.
Dim selection As String = Me.cboViewerType.Items(Me.cboViewerType.SelectedIndex).Text
Select Case selection
 Case "AcrobatReader"
  Me.arvWebMain.ViewerType = GrapeCity.ActiveReports.Web.ViewerType.AcrobatReader
 Case "HTML viewer"
  Me.arvWebMain.ViewerType = GrapeCity.ActiveReports.Web.ViewerType.HtmlViewer
 Case "RawHtml"
  Me.arvWebMain.ViewerType = GrapeCity.ActiveReports.Web.ViewerType.RawHtml
End Select
```

C#

```
// Set the type of the selected view in ViewerType property of the WebViewer.
{
    string selection = this.cboViewerType.Items(this.cboViewerType.SelectedIndex).Text;
    switch (selection) {
        case "AcrobatReader":
            this.arvWebMain.ViewerType =
GrapeCity.ActiveReports.Web.ViewerType.AcrobatReader;
            break;
        case "HTML viewer":
            this.arvWebMain.ViewerType =
GrapeCity.ActiveReports.Web.ViewerType.HtmlViewer;
            break;
        case "RawHtml":
            this.arvWebMain.ViewerType = GrapeCity.ActiveReports.Web.ViewerType.RawHtml;
            break;
    }
}
```

To migrate project that uses PDF Export

Re-create the application by referring the following basic operation or sample. Note that you don't need to re-create everything.
The basic difference between ActiveX viewer and PDF export is in the format of the data received from the web server to the client. An RDF file is generated on the Web server with ActiveX viewer and binary data of PDF format is created on the web server with PDF export. The other operations are common in ActiveX viewer and PDF export, there is no need to re-create the existing applications that are created using the ActiveX viewer. See Custom Web Exporting for more information.

Compatibility Guidelines

This topic guides you regarding the compatibility between the previous versions and ActiveReports 14. You may face issues related to following after the migration.

Show Method

In ActiveReports 6.0 and above, the Show method has been removed from the report class and as a result, the dependency to the main assembly (ActiveReports6.dll) and the viewer assembly (Viewer6.dll) is no longer available. Earlier, the main assembly and the viewer assembly in the execution environment were required to be added even for the applications that did not required preview. In ActiveReports 6 or above, you no longer need to add the viewer assembly.
If you are previewing reports using the following code in a previous version, you need to change it to the preview method that uses the Viewer component.

Before migration

Visual Basic

```
Private Sub Form1_Load(...) Handles MyBase.Load
 Dim rpt As New SampleReport
 rpt.Show()
End Sub
```

C#

```
private void Form1_Load(object sender, System.EventArgs e)
{
 SampleReport rpt = new SampleReport();
 rpt.Show();
}
```

In the following code, Form1 has been placed in the Viewer control. Here, if you set **Document** property of the report and then run the report, results obtained are similar to that with Show method.

After migration

Visual Basic

```
Private Sub Form1_Load(...) Handles MyBase.Load
 Dim rpt As New SampleReport
 Me.Viewer1.Document = rpt.Document
 rpt.Run()
End Sub
```

C#

```
private void Form1_Load(object sender, System.EventArgs e)
{
 SampleReport rpt = new SampleReport();
 this.viewer1.Document = rpt.Document;
 rpt.Run();
}
```

Print Method

In ActiveReports 7, the internal implementation of the Document.Print method has been changed. The Print method that was available in **Document** class upto ActiveReport 6 has been replaced by an extension method in **PrintExtension** class of Grapecity.ActiveReports namespace

(GrapeCity.ActiveReports.Viewer.Win.dll) introduced in ActiveReports 7. Therefore in ActiveReports 7 and above, you need to import namespace in order to use the Print method. For more details, see Print Methods.

Custom Toolbar

ActiveReports 7 onward, toolbar has been reformed to use ToolStrip class. Also, the Viewer feature has been added to the toolbar and as a result, the display order of toolbar buttons has also changed. For more details, see Customizing the Viewer Control.

In case you have set the toolbar as **Hidden**, modify the code as follows:

Before migration

Visual Basic

```
'Code for ActiveReports 6
Me.Viewer1.Toolbar.Tools(23).Visible = False 'Hide the [Annotation] button
Me.Viewer1.Toolbar.Tools(4).Visible = False 'Hide the copy button
```

C#

```
//Code for ActiveReports 6
this.Viewer1.Toolbar.Tools(23).Visible == false //Hide the [Annotation] button
this.Viewer1.Toolbar.Tools(4).Visible == false //Hide the copy button
```

After migration

Visual Basic

```
'Viewer1.Toolbar.ToolStrip.Items.RemoveAt(37)
'[Annotation] button is hidden in ActiveReports 7, we don't have to mention it.
 Viewer1.Toolbar.ToolStrip.Items.RemoveAt(5) 'Hide the copy button
```

C#

```
//Viewer1.Toolbar.ToolStrip.Items.RemoveAt(37);
//[Annotation] button is hidden in ActiveReports 7, we don't have to mention it.
 Viewer1.Toolbar.ToolStrip.Items.RemoveAt(5); //Hide the copy button
```

In case you have set additional buttons in the toolbar through the code, you need to replace it with the new API code.

Before migration (from Version 6)

Visual Basic

```
'Delete the existing Annotation button and add [Add Custom Annotation] button

Dim image As System.Drawing.Icon
image = New
System.Drawing.Icon(Me.GetType.Module.Assembly.GetManifestResourceStream("CustomAnnotations.NOTE16.ICO"))
Viewer1.Toolbar.Images.Images.Add(image)

Dim btn As DataDynamics.ActiveReports.Toolbar.Button
btn = New DataDynamics.ActiveReports.Toolbar.Button

btn.ButtonStyle = DataDynamics.ActiveReports.Toolbar.ButtonStyle.TextAndIcon

btn.ImageIndex = 14 'Add new image in Toolbar.Images
btn.Id = ToolIds.Annotation 'Set unique ID in the button
btn.Caption = "Custom Annotation"
btn.ToolTip = "Set confirmation mark"
Viewer1.Toolbar.Tools.RemoveAt(23) 'Delete the existing Annotation button.
Viewer1.Toolbar.Tools.Insert(23, btn)
```

```
Private Sub Viewer1_ToolClick(ByVal sender As Object, ByVal e As
DataDynamics.ActiveReports.Toolbar.ToolClickEventArgs) Handles Viewer1.ToolClick

'Describe about the event fired while pressing the annotation button

End Sub
```

C#

```
//Delete the existing Annotation button and add [Add Custom Annotation] button

System.Drawing.Icon image = default(System.Drawing.Icon);
image = new
System.Drawing.Icon(this.GetType.Module.Assembly.GetManifestResourceStream("CustomAnnotations.NOTE16.ICO"));
Viewer1.Toolbar.Images.Images.Add(image);

DataDynamics.ActiveReports.Toolbar.Button btn = default(DataDynamics.ActiveReports.Toolbar.Button);
btn = new DataDynamics.ActiveReports.Toolbar.Button();

btn.ButtonStyle = DataDynamics.ActiveReports.Toolbar.ButtonStyle.TextAndIcon;

btn.ImageIndex = 14; //Add new image in Toolbar.Images
btn.Id = ToolIds.Annotation;  //Set unique ID in the button
btn.Caption = "Custom Annotation";
btn.ToolTip = "Set confirmation mark";
Viewer1.Toolbar.Tools.RemoveAt(23);  //Delete the existing Annotation button.
Viewer1.Toolbar.Tools.Insert(23, btn);

private void Viewer1_ToolClick(object sender, DataDynamics.ActiveReports.Toolbar.ToolClickEventArgs e)
{
//Describe about the event fired while pressing the annotation button

}
```

After migration (from Version 6)

Visual Basic

```
'Delete the existing Annotation button and add [Add Custom Annotation] button

Dim image As System.Drawing.Icon
image = New
System.Drawing.Icon(Me.GetType.Module.Assembly.GetManifestResourceStream("CustomAnnotations.NOTE16.ICO"))

Dim btn As New ToolStripButton("Custom Annotation")
btn.DisplayStyle = ToolStripItemDisplayStyle.ImageAndText
btn.Image = image.ToBitmap
btn.ToolTipText = "Set confirmation mark"

'Viewer1.Toolbar.ToolStrip.Items.RemoveAt(37) 'The annotation button is by default hidden in ActiveReports
7, so it doesn't need description.
Viewer1.Toolbar.ToolStrip.Items.Add(btn)
'Viewer1.Toolbar.ToolStrip.Items.Insert(37,btn) 'Create event handler if you want to place button in a
specified location during the button click.
AddHandler btn.Click, AddressOf tsbAnnotation_Click

Private Sub tsbAnnotation_Click(sender As Object, e As EventArgs)

'Describe about the event fired while pressing the annotation button

End Sub
```

C#

```
//Delete the existing Annotation button and add [Add Custom Annotation] button
System.Drawing.Icon image = default(System.Drawing.Icon);
image = new
System.Drawing.Icon(this.GetType.Module.Assembly.GetManifestResourceStream("CustomAnnotations.NOTE16.ICO"));
ToolStripButton btn = new ToolStripButton("Custom Annotation");
btn.DisplayStyle = ToolStripItemDisplayStyle.ImageAndText;
btn.Image = image.ToBitmap;
btn.ToolTipText = "Set confirmation mark";
//Viewer1.Toolbar.ToolStrip.Items.RemoveAt(37); //The annotation button is by default hidden in
ActiveReports 7, so it doesn't need description.
Viewer1.Toolbar.ToolStrip.Items.Add(btn);
//Viewer1.Toolbar.ToolStrip.Items.Insert(37,btn); //Create event handler if you want to place button in a
specified location during the button click.
btn.Click += tsbAnnotation_Click;
private void tsbAnnotation_Click(object sender, EventArgs e)
{
//Describe about the event fired while pressing the annotation button
}
```

In C# code, remove the additional code of Viewer.ToolClick event handler in XXX.Designer.cs.

Control Borders

ActiveReports 6 or above does not provide support the use of control borders of the Line control. As a result, control border of the Line control set in the previous version will not get rendered after migrating to ActiveReports 7 or above.

You cannot set the control border in the following controls:

- Line
- PageBreak
- CrossSectionBox
- CrossSectionLine

HTTP Handler (Professional edition only)

In ActiveReports 2 or above, the upper case and lower case characters of hyperlink strings used for directly referring the report layout file have been clearly distinguished. This may affect the working after the migration if upper case and lower case characters are not used correctly.

WebViewer Control (Professional edition only)

There are certain modifications in the properties available within the property window. In ActiveReports 1, you can use Report property to specify the report to be displayed. In ActiveReports 2 or above **ReportName** property has been introduced to replace Report property.
Please note that the Report property is removed from the property window but not from the WebViewer object, and hence there is no change in the code.

The type to set the Report property and ReportName property are different.

- **Report ('Report Property' in the on-line documentation)** property: Object type
- **ReportName ('ReportName Property' in the on-line documentation)** property: String type

Migrate Execution Environment

In .NET and .NET Core frameworks, there is an assembly recognition mechanism with which the executable files or assembly files created in Visual Studio identify the assembly they depend. Therefore, it is necessary to have the same product version while building an application as that distributed in the run-time environment.

In case the product version is changed after distributing the application, the application will not run by simply changing the component (DLL file) in the run-time environment. To migrate the run-time environment of application to ActiveReports 14, migrate the development environment, migrate the project, and then re-create the application by rebuilding the solution. After doing, this you can deploy the created application along with ActiveReports 14 component.

Migrate from ActiveReports 2 COM

The reports in ActiveReports 2 COM, which is ActiveX version of ActiveReports, are saved in DSR/DSX format unlike .NET versions which saves report layouts in the XML based RPX format. Therefore, to migrate from ActiveReports 2 COM to ActiveReports for .NET, you need to save reports in ActiveReports 2 COM as .rpx file and embed it in ActiveReports 14 project. When you save the report layout as .rpx, ActiveReports only saves the code in the script editor. Any code behind the report is not saved to the RPX file, therefore, the code in the .cs or .vb file needs to be re-written.

Embedding an RPX file in a Visual Studio Project

> **Note:** Adding an ActiveReports 2 COM report layout file (.rpx) directly to a project is not supported. Be sure to migrate using the following steps.

1. From the Solution Explorer, select the project in which you want to add the item.
2. From the Project menu, select **Add New Item**.
3. From the Template pane, select ActiveReports 14 Section Report (code-based) or ActiveReports 14 Section Report (xml-based) file and click **Add**.
4. From the Report menu, select the **Load Layout** option.
5. Navigate to the location of the .rpx file, select the .rpx file, and click **Open**.

   > **Note:**
   > - Script codes such as Visual Basic and C# are not imported properly.
   > - The border shadow (Border.Shadow property) is not migrated.

ActiveReports 2 COM versus ActiveReports 14

This section explains about the difference between ActiveReports 2 COM (ActiveX version) and ActiveReports 14.

Comparison between Section Report Controls

The following table compares the controls available in Section Report in ActiveX product and ActiveReports 14 (.NET Core product).

ActiveX product	.NET Core product
Field	TextBox
Label	Label
CheckBox	CheckBox
Image	Picture
Line	Line
OleObject	OleObject

PageBreak	PageBreak
RichEdit	RichTextBox
Shape	Shape
SubReport	SubReport
ActiveX control (includes Barcode)	×（※1）
Frame	×（※2）
ADO Data Control	OLEDataSource（※3）
XML Data Control	XMLDataSource（※4）
RDO Data Control	×
DAO Data Control	×
—	Chart（※5）
—	CrossSectionBox（※6）
—	CrossSectionLine（※7）
—	ReportInfo（※8）
-	InputField （※**9**）

※1 It will only get migrated as basic class ARControl.

※2 Control placed inside the Frame will be migrated.

※3 Only Source (SQL) and ConnectionString properties set at design time are migrated.

※4 Only FileURL/RecordSetPattern property set at design time are migrated.

※5 Control that renders 2D/3D graph.

※6 Control that renders a rectangle across multiple sections.

※7 Control that renders a straight line across multiple sections.

※8 A control that renders the execution date and time of report, page number, and total number of pages in a specified format.

※ **9** A control that provides editable fields on exporting a report to PDF. This is a Professional Edition feature.

Important properties that have been changed in .NET Core product

	Property Name		
Control names in Section Report (.NET Core product)	ActiveX product	.NET Core product	Remarks
GroupHeader	GrpKeepTogether	GroupKeepTogether	The type itself is different
	Repeat	RepeatStyle	The type itself is different

CheckBox TextBox Label	WordWrap	WrapMode	The type itself is different
TextBox	DataValue	Value	
	SummaryDistinctField	DistinctField	
CheckBox	Alignment	CheckAlignment	The type itself is different
	Value	Checked	
Picture	Picture	Image	The type itself is different
Shape	Shape	Style	
SubReport	Object	Report	

Important properties added in .NET Core product

Control names in Section Report (.NET Core product)	Property Name	Description
GroupHeader	ColumnGroupKeepTogether	Make groups as one block on the same column
Detail	RepeatToFill	Adds an empty line
CheckBox TextBox Label	Padding	Sets the blank space inside a control
TextBox Label	CharacterSpacing	Sets the character pitch(interval) in points units
	LineSpacing	Sets the line spacing in point units
	TextJustify	Sets equal allocation
	VerticalText	Optimize character shape for vertical writing
	ShrinkToFit	Shrink the character according to the Control size
Picture	Description	Sets the visual explanation of Picture's appearance (valid when exporting Html)
Line	AnchorBottom	Draw the end of the ruled line till the adjacent section
Shape	RoundingRadius	Sets the roundness of round cornered rectangle in percentage unit
SubReport	CloseBorder	When a sub-report spans multiple pages, it sets whether to close it with a ruled line or not

The above table lists only those properties that are frequently used, although, in addition to the above properties, there are some newly added properties and changed properties in the .NET Core product.

Also, the properties and methods of OleObject control and RichTextBox (RichEdit) control have been significantly

changed. If you are using these controls, refer to "*Class Library Reference*" and adjust the settings as necessary.

Difference in Export Function

Export function of .NET Core product is developed to export the reports of .NET Core product.
Compared with the ActiveX products, various enhancements have been made in the Export function of .NET Core products along with the changes in properties and events.

For more information, please refer to the content below.

PDF Export

Subject	ActiveX product	.NET Core product
PDF Version	1.1〜1.3	1.1〜1.7
PDF image quality setting of internal image	JPGQuality property in settings (1〜100)	ImageQuality (three levels - Lowest, Medium, and Highest) and ImageResolution (75〜2400 dpi) property in settings
Non embedding of Japanese font	× (always embedded)	○ (※1, always embedded in Standard Edition)
Bold characters of Japanese font	× (output with normal thickness)	○ (※1)
Foreign letters	× (no output)	○ (※1)
Electronic Signature Time Stamp	×	○: Signature property (※1)
GIF and transparent elements of meta images	× (present in non-transparent state)	○
Output of bookmark	○: OutputTOCAsBookmarks property	○: ExportBookmarks property
Document property	×	○: Options property (※2)
Document display state	△: ShowBookmarksInAcrobat (only display bookmarks are configurable)	○: PdfDocumentOptions.DisplayMode property (※3)
Display window setting	×	○: PdfDocumentOptions property (※4)
UI setting	×	○: PdfDocumentOptions property (※5)
Print only	×	○: PdfDocumentOptions.OnlyForPrint property
Gets the state of progress	○: OnProgress event	×

※1 Can be set only for Professional Edition.

※2 Five items - Title, Author, Subject, Keywords, and Application can be set.

※3 Four types - None, Outlines, Thumbs, and FullScreen can be set.

※4 CenterWindow, FitWindow, and DisplayTitle can be set.

※5 Display permission of Menu bar (HideMenubar), Toolbar (HideToolbar), and UI Window (HideWindowUI) can be set.

Excel Export

Excel export (XlsExport class)

Subject		ActiveX product	.NET Core product
Save File	File version	Excel 95/97	Excel 95/97/2007 (OpenXML)
Layout	Space between cell and borders to prevent overlapping of cells	○：BorderSpace Property	×
	Column appearance in the output	○：DoubleBoundaries Property	×
	Generate page break automatically	○：GenPageBreaks property	△: (always output the page break)
	Display space between the report elements and the report margin	○：ShowMarginSpace property	×
	Print scale	○：SizeToFit property	×
	Remove vertical space	○：TrimEmptySpace property	○：RemoveVerticalSpace property
	Version	○：Version property	○：FileFormat property
	Display grid line	× (display fixing)	○：DisplayGridLines property
	Merge cells	×	○：UseCellMerging property
	Color palette	× (always keep palette in workbook)	○：UseDefaultPalette property
Printing	Paper size	× (dependent on excel default settings)	○：PageSettings.PaperSize property
	Page orientation	× (dependent on excel default settings)	○：PageSettings.Orientation property
Security	Read password	×	○：Security.Password property
	Write password	×	○：Security.WritePassword property
	Save as read only	×	○：Security.ReadOnlyRecommended property
	Username responsible to password protect	×	○：Security.ProtectedBy property

	an Excel sheet		
State of progress		○：OnProgress event	×

Excel export (SpreadBuilder API)

Subject		ActiveX product	.NET Core product
Implementation method		control in SpreadBuilder object	control in Workbook object
Save File	File version	Excel 95/97	Excel 95/97/2007（OpenXML）
	prior error check	○：GetSaveCaps method	×（processing due to Exception）
Layout	Cell merge/unmerge	×	○: Merge/UnMerge method
	Display grid line	× (always display)	○: DisplayGridLines property
	Line striking through a text	×	○: FontStrikeOut property
	Format setting by range specification	×	○: Use DDCells class
	Color palette	× (always keep palette in workbook)	○: UseDefaultPalette property
Printing	Page setup	Setting available in DDSheet object	Use settings in PageSetup class
	Black and white print	×	○: BlackAndWhite property
	Print without graphics	×	○: Draft property
	Scale print to fit page	○：SizeToFit property	○: FitToPage property
	Vertical scale page count	× (set to one page)	○: FitToPagesTall property
	Horizontal scale page count	× (set to one page)	○: FitToPagesWide property
	Scale factor	× (set to 100%)	○: Zoom property
	Paper size	× (depends on excel default settings)	○: PaperSize property
	First page number	× (automatic)	○: FirstPageNumber property
	Header height	×	○: HeaderMargin property
	Footer height	×	○: FooterMargin property
	Page orientation	× (set from 'left to right')	○: Order property
	Print setting of cell notes	× (set to 'none')	△: PrintNotes property ※only 'screen display image'

			can be set
Security	Read password	×	○: Password property
	Write password	×	○: WritePassword property
	Save as read only	×	○: ReadOnlyRecommended property
	Username responsible to password protect an Excel sheet	×	○: ProtectedBy property
	Workbook protection password	×	○: ProtectWorkbookPassword property

TIFF export

Subject		ActiveX product	.NET Core product
Compression scheme		Decided by the method to use	Setting available in CompressionScheme property
Compression format	none	×	○: "None"
	Lzw	×	○: "Lzw"
	Rle (PackBits)	○: Export/ExportTIFF method	○: "Rle" (however, default is black and white)
	CCITT Group3	○: FaxExportCITT3 method	○: "Ccitt3"
	CCITT Group4	○: FaxExport method	○: "Ccitt4"
Dithering		○: only FaxExport or FaxExportCITT3 methods can be used (※1)	○: Dither property (※2)
Resolution		○: only ExportTIFF method can be used	○: DpiX, DpiY property
State of progress		○: OnProgress event	×

※1 The white threshold can be set within the range of 0〜765.

※2 Valid only for "Rle", "Ccitt3", "Ccitt4". Also, this property is of Boolean type.

HTML export

Subject		ActiveX product	.NET Core product
File output	Image save location	○: AuxOutputPath property	× (always same path as HTML in output)
	Whether to embed CSS style inside	○: CreateCSSFile property	× StyleStream property

	the html file		
	Output file name	FileNamePrefix property setting	Specified by argument of Export method
	File output location	HTMLOutputPath property setting	Specified by argument of Export method
	JPEG compression ratio	○: JPEGQuality property	×
	Output in MHT format	○: MHTOutput property	×
	Output setting for multiple pages	MultiPageOutput property setting	MultiPage property setting
	Output of bookmarks	○: TableOfContents property (※1)	○: BookmarkStyle property (※2)
Layout	Insert code just before HEAD tag	○: HeadExtraInnerText property	×
	HTML version setting	○: HTMLVersion property	○: OutputType property
	Remove vertical space	×	○: RemoveVerticalSpace property
	Add HTML just before page output	○: ExportPageStart event	×
	Add HTML just after page output	○: ExportPageEnd event	×
State of progress		○: OnProgress event	×

※1 Either of these properties can be set - None, <DL>tag format, DHTML format, or XML format.

※2 Either of these properties can be set - None, HTML.

Text export

Subject	ActiveX product	.NET Core product
Encoding	△: Unicode property (※1)	○: Encoding property
Obtaining state of progress	○: OnProgress event	×

※1 Unicode format and ASCII format configurable only from either of them.

RTF export

Subject	ActiveX product	.NET Core product
Obtaining state of progress	○: OnProgress event	×

Difference in Barcode Control

The barcodes and their properties available in .NET Core product are different from that available in ActiveX product. While in ActiveX product, the barcode is rendered as an ActiveX control, in the .NET Core product, the Barcode is available as a control. For full list of barcodes available in ActiveReports, see Barcode topic.

Barcode formats

ActiveX product	.NET Core product	Remarks
CODE39	Code39, Ansi39	※1
CODE39(Full ASCII)	Code39x, Ansi39x	
CODE49	Code49	
CODE93	Code_93	Only Uppercase of letters,%, $, *, ., +, -, Space, number can be set.
	Code93x	whole letter of ASCII letter set can be set
CODE128	Code_128auto	※2
	Code_128_A	CODE-A fixed format
	Code_128_B	CODE-B fixed format
	Code_128_C	CODE-C fixed format
JAN8	EAN_8	
JAN13	EAN_13	
EAN128	UCCEAN128	※2, 3, 4
	EAN128FNC1	※2, 3, 4
ITF	Code25intlv	※5, 6
POSTNET5, 9, 11	PostNet	without depending on data number of digits, symbol can be generated
UPC/A	UPC_A	
UPC/E	UPC_E0, UPC_E1	※7
UPC/E Addon2, Addon5	UPC_E0, UPC_E1	
NW-7(CODABAR)	Codabar	
Customer Barcode	JapanesePostal	

※1 Code39 (Code39x) and Ansi39 (Ansi39x) have the same bar code configuration specifications, but the default width ratio of narrow bar and wide bar is different. The former is 1: 2, whereas the latter is 1: 3.

※2 Start character setting of CODE128 is different for ActiveX product and .NET Core product.

※3 In CODE128 format, there is no function to arbitrarily switch code sets.

※4 In CODE128 (EAN128) format, there is no function to insert FNC2 to FNC4 in arbitrary place.

※5 Check digit is not calculated automatically. For .NET Core products, you need to set a value with check digit added.

※6 Bearer bar is not supported.

※7 UPC_E0 is compressed zero type of UPC symbol and can only set only numbers. UPC_E1 is generally used in retail shops price labels and it supports six numeric characters.

Important ActiveX product properties that are not supported by .NET Core product

The properties that do not exist in .NET Core product but affect generated barcode are listed as follows:

Property name	Description
BarRatio	Set bar ratio. We can set relative width ratio of fine module to thick module in the range of 1:2 to 1:3 using NWRatio property.
LongModuleSize	Set thick module width.
TargetDpiX	Set the resolution of screen in the horizontal direction.
TargetDpiY	Set the resolution of screen in the vertical direction.

About ActiveReport Object

The ActiveReport object (the base of ActiveX product report), included not just the engine and the report layout information, but also the viewer function. The .NET Core product provides SectionReport class (the base class of section report) and the viewer function, and the Viewer class is reconfigured.
Also, In ActiveX product the generated page information was stored in the Pages class but in .NET Core product it has been reconfigured to the Document class and PagesCollection class. Along with these configuration updates, some methods and events have also been modified. The following table lists the important changes:

Difference in property

ActiveX product	.NET Core product		Remarks
	Class	Property name	
AllowSplitters	Viewer	AllowSplitter	
documentName	SectionDocument	Name	
PageBorder	×	×	※1
Pages	SectionDocument	Pages	※2
Printer	SectionDocument	Printer	
RulerVisible	×	×	Ruler does not exist.
ShowMessages	×	×	※3
ScriptDebuggerEnabled	SectionReport	EnableScriptDebugging	
Status	SectionReport	State	
TOC	SectionDocument	Bookmarks	
TOCEnabled	TOCPanel (Viewer.Sidebar)	Enabled	
TOCVisible	TOCPanel (Viewer.Sidebar)	Visible	※4
TOCWidth	Sidebar(Viewer)	Width	
Toolbar	Viewer	Toolbar	
ToolbarVisible	ToolStrip (Viewer.Toolbar)	Visible	
Zoom	Viewer	Zoom	

※1 Draw border in CrossSectionBox control and other controls.

※2 In ActiveX product, it is the collection of Canvas objects but in .NET Core product, it is the collection of Document.Page.

※3 If an exception is thrown due to an error in the .NET Core product, use Try and Catch Statement.

※4 Use the ToggleVisibility method of the Viewer.Sidebar object to display headings.

Difference in method

ActiveX product	.NET Core product		
	Class	Method name	Remarks
Export	(each export filter)	Export	
PageSetup	×	×	※1
PrintReport	SectionDocument PageDocument Viewer	Print	
Refresh	×	×	※2

※1 Implemented using PageSetupDialog Class.

※2 Reset the report document using Document property and LoadDocument method.

Difference in event

ActiveX product	.NET Core product		
	Class	Event name	Remarks
Error	×	×	※1
FindProgress	Viewer	Find	
HyperLink	Viewer	HyperLink	
PromptDialogClosed	SectionReport	ParameterUIClosed	
TOCClick	Viewer	TableOfContentsClick	
TOCSelChange	Viewer	TableOfContentsSelectedIndexChanged	
ToolbarClick	×	×	※2

※1 If an exception is thrown due to an error in the .NET Core product, use Try and Catch Statement.

※2 Implement using Viewer.Toolbar.ToolStrip class. For more information please see Customize the Viewer Control.

Coexistence of ActiveReports Designers

ActiveReports 14 can be installed and used on the same machine in which other older versions of ActiveReports for .NET (except ActiveReports 13) have been installed.

Compatibility of ActiveReports designer with Visual Studio

ActiveReports designer of different versions can be used by integrating the designer with Visual Studio IDE (Integrated development environment). The following table shows the Visual Studio versions corresponding to the ActiveReports designer versions that are supported.

	VS .NET 2002	VS .NET 2003	VS 2005	VS 2008	VS 2010	VS 2012	VS 2013	VS 2015	VS 2017	VS 2019
ActiveReports 1	○	○	×	×	×	×	×	×	×	×
ActiveReports 2	×	○	○	×	×	×	×	×	×	×
ActiveReports 3	×	○	○	×	×	×	×	×	×	×
ActiveReports 6	×	×	○	○	○	×	×	×	×	×
ActiveReports 7	×	×	×	○	○	○	○	×	×	×
ActiveReports 8	×	×	×	○	○	○	○	×	×	×
ActiveReports 9	×	×	×	×	○	○	○	○	×	×
ActiveReports 10	×	×	×	×	○	○	○	○	×	×
ActiveReports 11	×	×	×	×	○	○	○	○	○	×
ActiveReports 12	×	×	×	×	×	○	○	○	○ (VS SP2+)	×
ActiveReports 13	×	×	×	×	×	○	○	○	○ (VS SP2+)	○
ActiveReports 14	×	×	×	×	×	×	○	○	○ (VS SP2+)	○

To switch the designers of versions prior to ActiveReports 14, use the switcher tool. See ActiveReports 13 help file for more information.

Import Reports

This section explains about importing reports in ActiveReports using the ActiveReports Import Wizard.

Topic	Content
Importing Crystal/MS Access Reports	This section describes about importing Crystal and MS Access reports in ActiveReports.

Import Excel	This section describes about importing MS Excel reports in ActiveReports.
Import RPX	This section describes how to import ActiveReports section reports to ActiveReports Page or RDL reports.

Import Crystal Reports/MS Access Reports

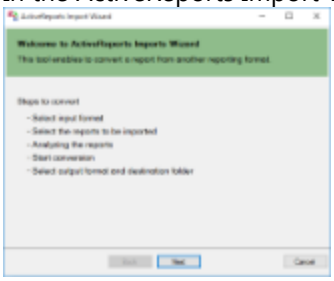
You can import a Crystal Reports report, a Microsoft Access report, an Excel file, or an ActiveReports section report in ActiveReports by running the ActiveReports Import Wizard.

To learn about importing Excel files, please see Importing Excel.

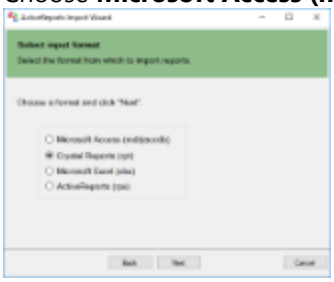
To learn about importing ActiveReports section reports, please see Importing RPX.

Importing Crystal reports/MS Access Reports in the ActiveReports Import Wizard

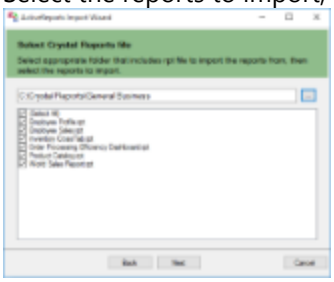
1. Run the ActiveReports Import Wizard. The wizard can be run from the start menu or by executing GrapeCity.ActiveReports.Imports.Win.exe from **C:\Program Files (x86)\GrapeCity\ActiveReports 14\Tools** location.
2. In the ActiveReports Import Wizard that appears, click **Next**.



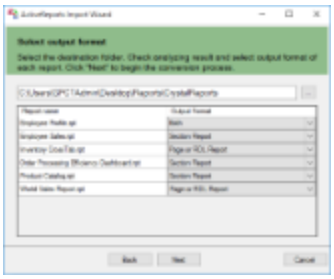
3. Choose **Microsoft Access (mdb or accdb)** or **Crystal Reports (rpt)** as the input format and click **Next**.



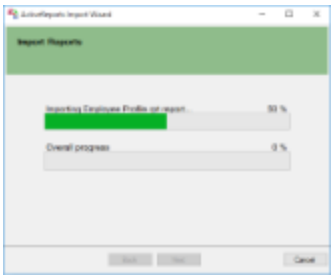
4. Click the ellipsis button to browse to the location that contains the files that you want to import. A list of files that you can import appears.
5. Select the reports to import, click **Open,** and then click **Next** to analyze them.



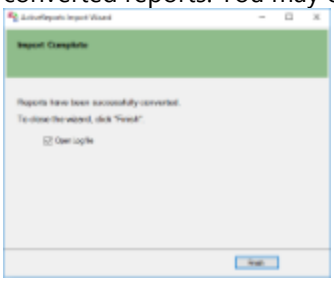
6. Use the ellipsis button to select a destination folder to store the converted reports. Also select an output format (Section Report, Page Report or RDL Report or Both) for each report in the Output Format column.



7. Click **Next** to start the conversion.



8. Once the conversion process is complete, click **Finish** to close the wizard and go the destination folder to view the converted reports. You may optionally leave the check on for the **Open Log file** checkbox to see the results log.



The import wizard converts reports to the closest possible ActiveReports format, but due to differences between products and versions, the extent to which your reports are converted depends on your specific report layout. You may have to partially redesign the report and add script or code to get the same output as Microsoft Access Reports or Crystal Reports.

When converting to Page Reports or RDL Reports, whether a report is imported as a Page Report or RDL Report, depends on the following factors:

- If a report has a single detail section it is imported as a Page Report.
- If a report has a CrossTab control and its layout is composed of multiple sections it is imported as an RDL Report.

> **Note:** Sections in a report appear as BandedList.

Please refer to the additional information below, to understand the conversion process in detail.

Importing Crystal Reports

To import Crystal Reports in ActiveReports, you need to install Visual Studio and Crystal Reports for Visual Studio on your machine. The supported versions of Visual Studio and corresponding Crystal Reports are as follows:

Visual Studio	Editions	Crystal Reports	Assembly Version

2008	Professional, Team System	Crystal Reports for Visual Studio 2008	10.5.3700.0
2010, 2012, 2013, 2015, 2017, 2019	...	SAP Crystal Reports, developer version for Microsoft Visual Studio	13.x.x.x

Crystal Report controls are converted in ActiveReports as follows:

Crystal Report	Section Report	Page Report/RDL Report	Note
Box	Shape	Container	The LineWidth property and rounded boxes are not imported. If the Box control extends to multiple sections, the box is imported as line controls.
CrossTab	SubReport	BandedList	CrossTab control is not imported as it is.
Line	Line	Line	The size of Dot and Dash (the LineStyle property) is not the same as the original report.
Subreport	SubReport	Subreport	Set the subreport in code after conversion.
TextObject	Label	Textbox	Only page number, total page, page n of m in Special Fields are imported.
FieldObject	TextBox	Textbox	Only page number, total page, page n of m in Special Fields are imported.
Picture	...	Container	Picture object is not converted.

Importing Microsoft Access Reports

To import Microsoft® Access® reports in ActiveReports, you must have Access 97, 2000, 2002, 2003, 2007, 2010, or 2013 installed on your system.

Microsoft Access report controls are converted in ActiveReports as follows:

Microsoft Access Report	Section Report	Page Report/RDL Report	Note
Rectangle	Shape	Container	Controls placed inside the Rectangle control are also imported along with the parent control.
CheckBox	Label	Textbox	...
Image	...	Image	Image control is not converted while converting to a Section Report.
Label	Label	Textbox	...
Textbox	TextBox	Textbox	...
Line	Line	Line	...
Page Break	PageBreak	Container	In Page Reports and RDL Reports, the **PageBreakAtEnd** property is automatically set to True on importing a Page Break control.
Subform/Subreport	SubReport	Subreport	...

Limitations in Crystal Report/MS Access conversion

- Any controls, functions, and text formats which are not supported by ActiveReports are not imported.
- The shadow property of a control is not imported while converting a report.
- The OLE object is not imported in ActiveReports as it is treated as PictureObject in the object structure of Crystal Reports.
- In Microsoft Access reports, VBA code appears in as commented statements in script. You have to modify the code after importing.

Import Excel

The migration from Microsoft Excel file to ActiveReports can now be accomplished by using the **ActiveReports Import Wizard**. The ActiveReports Import Wizard is particularly useful when you want to convert multiple sheets of an Excel file to ActiveReports. It saves the time and effort of a developer to manually replicate the layout of each sheet of an Excel file in ActiveReports.

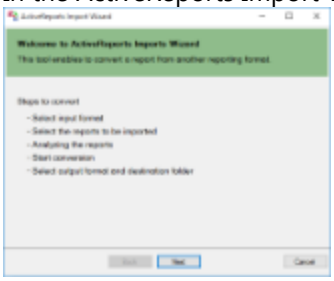
You can import a single sheet or multiple sheets of an Excel file to a Page or an RDL report with just a few clicks. A single Excel sheet is imported as a report file, and the report name is the name of the sheet. An Excel file with multiple sheets is by default imported as separate report files, and the report names are the name of the corresponding sheets in the Excel file. You can also set **Merge all sheets into a single report file** option in the ActiveReports Import Wizard to import multiple sheets of Excel file as different pages of the report.

- **Importing Excel files in the ActiveReports Import Wizard**
- **Defining Table area in an Excel file**
- **Naming Rules for defining a Table area in Excel**
- **Conversion Rules for Table area in Excel**
- **Supported Objects and Properties**
- **Limitations**

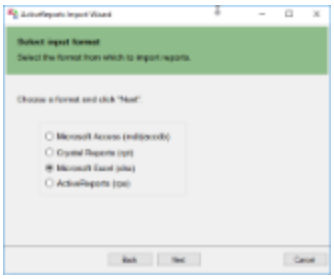
> **Note:** The import formats that are not supported are .xls (Excel 97-2003) and .xlsm (Open XML with macro).

Importing Excel files in the ActiveReports Import Wizard

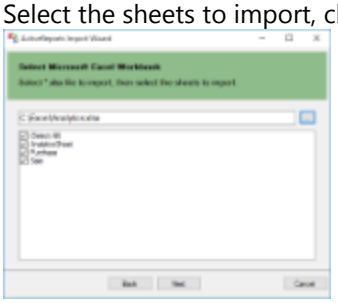
1. Run the ActiveReports Import Wizard. The wizard can be run from the start menu or by executing GrapeCity.ActiveReports.Imports.Win.exe from **C:\Program Files (x86)\GrapeCity\ActiveReports 14\Tools** location.
2. In the ActiveReports Import Wizard that appears, click **Next**.



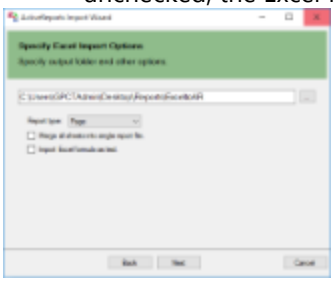
3. Choose **Microsoft Excel (xlsx)** as the input format and click **Next**.



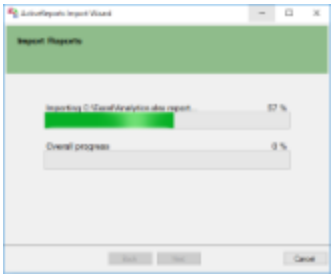
4. Click the ellipsis button to browse to the location that contains the files that you want to import. A list of files that you can import appears.
5. Select the sheets to import, click **Open,** and then click **Next** to analyze them.



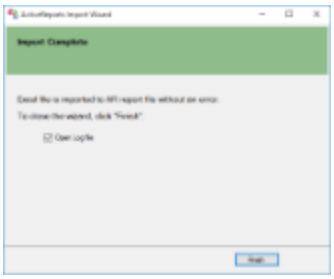
6. Use the ellipsis button to select a destination folder to store the converted reports. You can set the following options:
    - **Report Type:** Choose from Page report or RDL report formats to import the Excel file. Note that Page report does not support multiple data sources. You should select RDL report type if you want to add multiple data sources to the report.
    - **Merge all sheets into single report file:** Choose this option to import sheets of the Excel file as separate pages of a Page report. The report name is the name of the first sheet of the Excel file.
    - **Import Excel formula as text:** Choose this option to import Excel formula as text. If you keep the option unchecked, the Excel formula is imported as a calculated result.



7. Click **Next** to start the conversion.



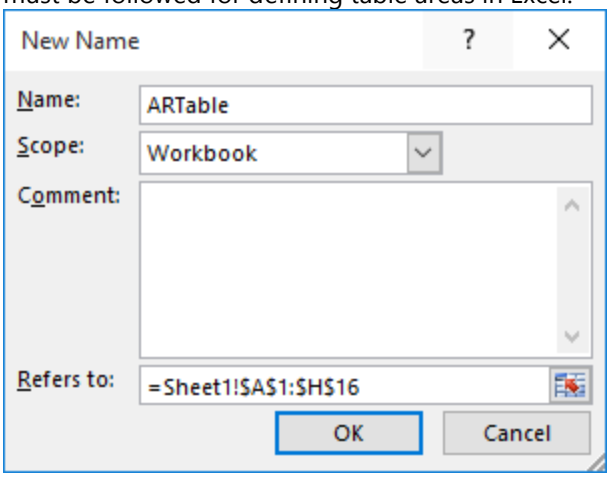
8. Once the conversion process is complete, click **Finish** to close the wizard and go the destination folder to view the converted reports. You may optionally leave the check on for the **Open Log file** checkbox to see the results log.



Defining Table area in an Excel file

Table area in Excel is the range of cells representing a Tabular data in the Excel.
If an Excel file has the table area that you want to import into ActiveReports as the Table data region, you must define the Table area first and then run the ActiveReports Import Wizard. Otherwise, defining the table area is not required.

1. Open the Excel file and select the table area.
2. Right-click to view the context menu.
3. Select the **Define Name** option.
4. In the **New Name** dialog box, define the table area and the rows based on **Naming Rules**. These naming rules must be followed for defining table areas in Excel.



5. Click OK.

Naming Rules for defining a Table area in Excel

To obtain the required table sections in ActiveReports' Table data region, you need to define the table area and its rows in the Excel file. In general, the table area is defined as **ARTable#.******, where:

- **#** is used to define more than one table areas. It can be any character, except symbol or character restricted by Excel. For example, ARTable, ARTable_1, or ARTableAbc.
- ****** is the name of the row (section): Detail, TableHeader, TableFooter, GroupHeader, or GroupFooter.
  In case of multiple rows (Table Header/Footer, Detail, Group Header/Footer), you need to set "#" for each row as well (for example, GroupHeader1, GroupHeader2, etc.)

Example 1: **To define a single table area**

Action	Naming Rule
Define whole table area	ARTable
Define each row	ARTable.Detail ARTable.TableHeader ARTable.TableFooter ARTable.GroupHeader1 ARTable.GroupFooter1

Example 2: To define a multiple table area

Action	Naming Rule
Define whole table area	ARTable1 ARTable2
Define each row	ARTable1.Detail ARTable1.TableHeader ARTable1.TableFooter ARTable2.Detail ARTable2.TableHeader ARTable2.TableFooter

Conversion Rules for Table area in Excel

The table area of Excel is imported as a Table data region in ActiveReports based on the following conversion rules.

- If the defined table area of Excel has three or more rows, the file data is converted to Table Header, Detail, and Table Footer as:

Excel Table	ActiveReports Table
Top Row	Table Header
Bottom Row	Table Footer
Other Rows	Detail

Note: For the Table Detail row, values for properties such as Value, Location, Size, etc. are imported from the cells of the first row.

- If the defined table area of Excel has two rows, the file data is converted as follows.

Excel Table	ActiveReports Table
First Row	Table Header
Second Row	Detail

- If the defined table area of Excel has only one row, the file data is converted as follows.

Excel Table	ActiveReports Table
First Row	Detail

Supported Objects and Properties

Excel		Page/RDL Report	
Item	Property	Item	Property
Page	Page setting	**Report**	-
	Size		PaperSize
	Orientation: Portrait		PaperOrientation: Portrait
	Orientation: Landscape		PaperOrientation: Landscape
	Margins (Top, Bottom, Left, Right)		Margins (Top, Bottom, Left, Right)
Cell	Value		Value
	Location		Location (Left, Top)
	Size		Size (Width, Height)
	Alignment		-
	Horizontal alignment: General		TextAlign: General
	Horizontal alignment: Left (Indent)		TextAlign: Left
	Horizontal alignment: Center		TextAlign: Center
	Horizontal alignment: Right (Indent)		TextAlign: Right
	Horizontal alignment: Justify		TextAlign: Justify
	Horizontal alignment: Distributed (Indent)		TextJustify: DistributeAllLines
	Vertical alignment: Top		VerticalAlign: Top
	Vertical alignment: Center		VerticalAlign: Middle
	Vertical alignment: Bottom		VerticalAlign: Bottom
	Text control: Wrap text		WrapMode: WordWrap
	Text control: Shrink to fit		ShrinkToFit: True
	Text direction: Left-to-Right		Direction: LTR
	Text direction: Right-to-Left		Direction: RTL
	Font		-

	Name	**TextBox**	FontFamily
	Style: Regular		FontStyle: Normal
	Style: Italic		FontStyle: Italic
	Style: Bold		FontWeight: Bold
	Style: Bold Italic		FontWeight: Bold
	Size		FontSize
	Color		Color
	Underline: None		TextDecoration: None
	Underline: Single		TextDecoration: Single
	Border		-
	Line style (Top, Bottom, Left, Right): xlLineStyleNone		BorderStyle: None
	Line style (Top, Bottom, Left, Right): xlContinuous		BorderStyle: Solid
	Line style (Top, Bottom, Left, Right): xlDot		BorderStyle: Dotted
	Line style (Top, Bottom, Left, Right): xlDash		BorderStyle: Dashed
	Line style (Top, Bottom, Left, Right): xlDouble		BorderStyle: Double
	Line color		BorderColor
	Line weight: xlThin		BorderWidth: 1pt
	Line weight: xlMedium		BorderWidth: 2pt
	Line weight: xlThick		BorderWidth: 3pt
	Fill		-
	Background color		BackgroundColor
Table area	Each cell in a table area is converted to TextBox report item. **Note:** Whole table area is imported in ActiveReports even if table data is filtered.	**Table**	Location (Left, Top) Size (Width, Height) FixedSize (Width, Height)
Picture	Picture object is converted to Image report item.	**Image**	Value Source: Embedded Sizing: FitProportional Location (Left, Top) Size (Width, Height)

Limitations

- An Excel file that contains merged cells and table areas that are partially out of bounds, is not imported.
  - **Merged cell** - If merged cell starts within the page bounds and ends outside of them, the cell is not imported.

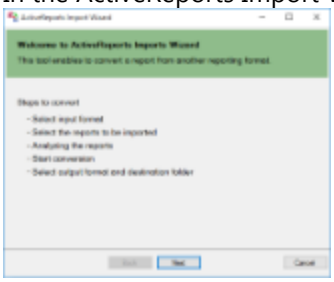
    - **Table area** - If table area starts within the page bounds and ends outside of them, the table area is not imported.
- ActiveReports Import Wizard does not support conversion of a **password protected Excel** file.
- The layout of only the first page of an Excel, as shown in the Page Break Preview option, is imported.
- Vertical texts can not be imported.
- The following Excel items are not imported to ActiveReports.
    1. **Page**
        - Header/Footer
    2. **Cell**
        - Number format (all categories, like Number, Date, Currency, etc.)
        - Strikethrough
        - Border (diagonal)
        - Fill effect
        - Fill pattern
        - Comment
        - Hyperlink
        - Table styles
        - Conditional formatting
    3. **Object**
        - Table
        - Shape
        - Chart
        - Pivot Table
        - WordArt
        - ClipArt

Import RPX

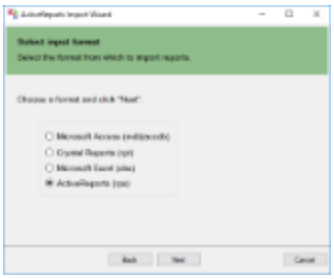
The ActiveReports Import Wizard helps you to convert existing ActiveReports Section report(s) (rpx) to Page and RDL reports.

Importing Section Report(s) (rpx) in the ActiveReports Import Wizard

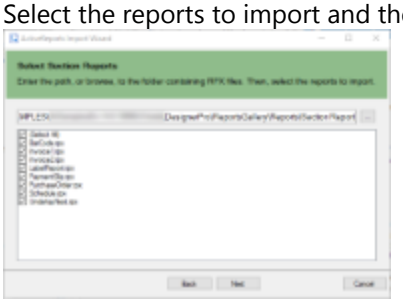
1. Run the ActiveReports Import Wizard. The wizard can be run from the start menu or by executing GrapeCity.ActiveReports.Imports.Win.exe from **C:\Program Files (x86)\GrapeCity\ActiveReports 14\Tools** location.
2. In the ActiveReports Import Wizard that appears, click **Next**.



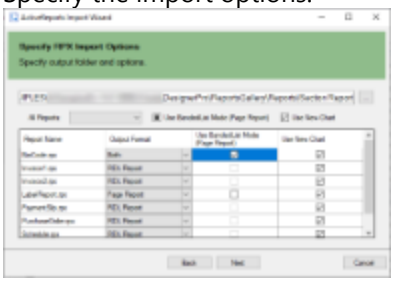
3. Choose **ActiveReports (rpx)** as the input format and click **Next**.



4. Click the ellipsis button to browse to the location that contains section report(s).
5. Select the reports to import and then click **Next** to analyze them.

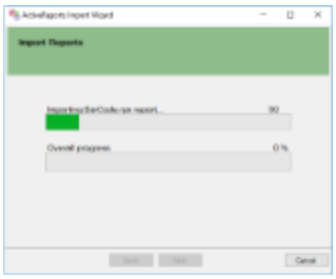


6. Specify the import options.

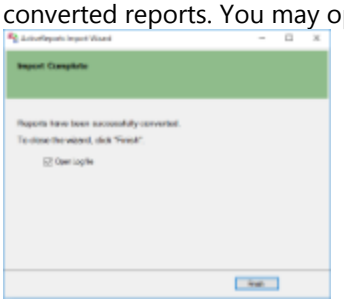


    1. Use the ellipsis button to select a destination folder to store the converted reports.
    2. Select the output format of the reports.
       From the drop-down, select output format as Page, RDL, or Both. If 'Both' is selected, then the converted Page and RDL reports are added with postfixes _page and _rdl, respectively in the destination folder.
    3. Select whether to use Banded List for Page report.
       The option to select **Use Banded List Mode** is available if the output format of a report is Page report.
        - If you want all controls to be placed on one BandedList in the same way as in RDL report, you should use banded list mode. In this case, the Fixed size for Page report needs to be set manually.
        - If you want to convert a Section report that has only one section (that is detail), you should not use banded list mode. With 'Use Banded List Mode' not selected (default):
            - controls are not placed on BandedList
            - sections such as Report Header/Report Footer and Page Header/Page Footer are ignored
            - all converted controls are placed on one Page and treated as they have been on one section

       The option to select **Use New Chart** lets you convert an existing chart to a new chart. By default, this option is selected; if unselected, the chart converts to classic chart.
7. Click **Next** to start the conversion.



8. Once the conversion process is complete, click **Finish** to close the wizard and go the destination folder to view the converted reports. You may optionally leave the check on for the **Open Log file** checkbox to see the results log.



> ⚠ **Important:** At conversion, any script from the script editor is ignored.

Section report controls are replaced with the RDL or Page report controls as follows.

Section Report Control	RDL/Page Report Control
Label	TextBox
TextBox	TextBox
CheckBox	CheckBox
RichTextBox	FormattedText
Shape	Shape
Picture	Image
Line	Line
Barcode	Barcode
SubReport	SubReport
Chart	Chart (New or Classic) (See the section **Conversion Limitations for Chart Control to New Chart**.)
ReportInfo	TextBox
CrossSectionLine	Line (in Page report when BandedList is not used)
CrossSectionBox	Container (in Page report when BandedList is not used)
PageBreak	Container in RDL report with 'Height=0in' and 'PageBreakAtEnd=True' Not supported in Page report

Conversion rules for Section report to Page report

- The converted report inherits these properties of Section report - Paper Size, Orientation, and the Margins.
- If controls are placed outside the paper size (red line appears), they are not converted.
- If the sum of height of all sections (PageHeader, Detail, etc.) in a Section report is more than its paper height, controls above or below the paper height are ignored.

Conversion rules for Section report to RDL report

- The entire Section report is converted to an RDL report as the BandedList control.
- PageHeader and PageFooter sections are automatically created at conversion. You must not remove these sections even if no content is available for them.

Conversion Limitations (both Page and RDL)

- The support of **Chart**, **Subreport**, and **RichTextBox** controls is limited to the basic functionality.
- Unused database fields are not imported.
- Calculated fields are converted into simple expressions.
- **PageTotal** and **PageCount** summary functions are not supported.
- **Visual Basic functions** are not supported. Expressions with function calls are imported as is.
- Following properties of Section report, on conversion to Page/RDL report, are not supported.
  - Culture
  - DataMember
  - ExpressionErrorMessage
  - MaxPages
  - TrayHeight
  - TrayLargeIcon
  - UserData
  - Watermark (for RDL and Page (using banded list mode))
- MaxLength property for RichTextBox control on conversion to Page/RDL report is not supported.

Conversion Limitations for Chart control to New Chart

- **BezierXY**, **Bubble**, **HiLo**, **HiLo Open Close**, and **LineXY** chart types are not properly implemented.
- **Axis labels** may be duplicated when using several **Series** with the same category grouping.
- **Colors** may differ when set to default.
- **Marker labels** may disappear when Field type doesn't match Format type and implicit type conversion is required.
- **Custom angle** for labels are not supported (can be only fixed 0, 90, 270).
- **Multiple chart titles** are not supported (can be only one title).
- **Stacked** option for Charts is not supported.

Report Types

ActiveReports provides a number of ways to design a report. In this section, learn about the choosing a report type based on your layout requirements. Depending on the type of report you select, you also get various file formats to create your reports.

Topic	Content

Page Report	The new Page report offers you a way to create very specific styles of reports, ideal for duplicating legacy paper forms.
Report Definition Language (RDL) Report	Learn how to build interactive reports with unique features where controls can grow and shrink.
Code-Based Section Report	Create reports based on C# or Visual Basic in Visual Studio.
XML-Based Section Report	Create reports based on XML layout in Visual Studio.

Report Layout Types

You can design reports using different layouts depending on your requirements. This section introduces these layout types and describes the differences between them to allow you to select the one that suits your report.

Page Layout

In a **Page Layout**, you design reports at the page level without any banded sections. This lets you place controls anywhere on the report.

In a Page report, controls do not change in size based on the data, but you can use an Overflow Place Holder to handle any extra data.

RDL Layout

In a RDL report, controls grow vertically to accommodate data.Controls can grow and shrink, you can set up interactive sorting, you can set up drill-down reports in which detail data is initially hidden, and can be toggled by other items, and you can add drill-through links to other reports and to bookmark links within reports.

Section Layout

In a **Section Layout**, you design reports in banded sections. A PageHeader, Detail and PageFooter section appear by default, and you can remove any but the detail section. Right-click the report and select **Insert** to add other section pairs like ReportHeader and ReportFooter, or GroupHeader and GroupFooter.

A report section contains a group of controls that are processed and printed at the same time as a single unit. All sections except the detail section come in pairs, above and below the detail section. When you use group headers and footers, the detail section processes for each group, and then the next group processes a group header, related details, and group footer. See Grouping Data for more information.

You can hide any section that you do not want shown by setting the **Visible** property of the section to **False**.

Report File Format Types

You can create reports in a number of file formats with a varied set of features. This section describes the use of each of these file formats.

Report Template Formats

To create a report, a user must select one of the following templates containing the report layout. See Quick Start and Install ActiveReports topics for details on available report templates and how to access these.

- **RDLX:** This is an XML-based proprietary file format that provides custom extensions to the Report Definition Language (RDL) files used by SQL Server Reporting Services. These are stand-alone files that you can process without compiling them into your application. You can customize the report through the Script Tab by embedding

script in the report.
See this msdn page for more on RDL report.

- **VB** or **CS**: These are code-based reports, and are saved as C# or Visual Basic files that are compiled into your applications. They have corresponding code views similar to Windows forms and provide a design and coding experience in line with Visual Studio. This format is ideal for developers who are comfortable with coding in .NET programming languages and would like to use the extensive event-based API provided by ActiveReports in the code-behind rather than design view. You may also use the scripts in the Script Tab instead of the code behind.
- **RPX**: This is an XML-based proprietary file format that the ActiveReports engine can process without compiling it into an application. Instead of Visual Basic or C# code behind, you can customize the report with script embedded in the report XML using the Script Tab. You can also use an RPX file with script as a stand-alone file in a Web project.

Additional File Formats

ActiveReports also provides some additional file formats for reports. Each of these formats is used for a specific purpose as described below.

- **RDLX-master:** This is a master report file that you can reference from other RDLX report files for a standard layout, for example, you can add company logo and address sections. This file is loaded each time the report is executed, so you can change the logo on all of your reports by just changing it on the master report.
- **RDLX-theme:** This is a theme file that consists of a collection of styles that you can apply to a report. See Themes for further details.
- **RDSX:** This is a proprietary format that is created when you share a data source, making it available to multiple reports.
- **RDF:** This is the Report Document Format, in which the data is static. You can save a report in this format to display the data that is retrieved. Once a report has been saved to an RDF file, it can be loaded into the viewer control. See Save and Load RDF Report Files for further details.

See the following list of file formats available in each layout.

Format	Page Layout/RDL Layout	Section Layout
RDLX	✓	✘
VB or CS	✘	✓
RPX	✘	✓
RDLX-Master	✓	✘
RDLX-Theme	✓	✘
RDSX	✓	✘
RDF	✘	✓

Features comparison between report types

In ActiveReports, the features available in a report depend on the type of report you select. See the following comparison list of features with each report type:

Feature	Section report	Page report	RDL report
Viewers & Editors			
Visual Studio Integrated Designer	✓	✓	✓
Expressions Editor	✘	✓	✓
Designer Script Editor	✓	✓	✓
Windows Form Viewer	✓	✓	✓
WebViewer (Pro Edition). Includes viewer types HTML, RawHTML, and PDF.	✓	✓	✓
HTTP Handlers (Pro Edition)	✓	✓	✓
Report Controls			
BandedList	✘	✓	✓
List	✘	✓	✓
Tablix	✘	✓	✓
Table	✘	✓	✓
OverflowPlaceHolder	✘	✓	✘
Chart	✓	✓	✓
Barcode	✓	✓	✓
Bullet	✘	✓	✓
CheckBox	✓	✓	✓
Container	✘	✓	✓
CrossSectionLine	✓	✘	✘
CrossSectionBox	✓	✘	✘
FormattedText	✘	✓	✓
Image	✘	✓	✓
Label	✓	✘	✘
Line	✓	✓	✓
PageBreak	✓	✘	✘
Picture	✓	✘	✘
ReportInfo	✓	✘	✘
RichTextBox	✓	✘	✘
Shape	✓	✓	✓
Sparkline	✘	✓	✓

SubReport	✓	✓	✓
TextBox	✓	✓	✓
TableOfContents	✘	✓	✓
Interactivity			
Hyperlinks	✓	✓	✓
Parameters	✓	✓	✓
Drill through	✘	✓	✓
Drill down	✓	✓	✓
Filtering	✘	✓	✓
Grouping	✓	✓	✓
Sorting	✘	✓	✓
Data Connections			
Standard Data Sources supported (e.g. SQL, OleDB, XML)	✓	✓	✓
Unbound Data Source	✓	✓	✓
Shared Data Source	✘	✓	✓
Export			
Export Filters	✓	✓	✓
Rendering Extensions	✘	✓	✓
PDF advanced export features: digital signatures, time stamp, bold font emulation (Pro Edition)	✓	✓	✓
Miscellaneous			
Master Reports	✓	✘	✓
Themes	✘	✓	✓
Collation	✘	✓	✓
Styles (through Report Settings dialog)	✓	✘	✘
Printing	✓	✓	✓
Stand-alone Applications			
ActiveReports Viewer	✓	✓	✓
ActiveReports Theme Editor	✘	✓	✓
ActiveReports Designer (stand-alone application)	✓	✓	✓

Page Report

The new Page report offers you a way to create very specific styles of reports that are very difficult, if not impossible, in other .NET reporting tools. You design this type of report on a page where none of the report controls can grow or shrink at run time, making it ideal for duplicating legacy paper forms.

As with all Page reports, instead of report sections where you place report controls, you place data regions and controls directly on the page. But with Page reports, there is no need to use code or add measurements to make sure that everything fits. Unlike the RDL Report, the controls remain fixed at run time, so you can drop a table on the report, set a property to size it exactly how you want it, and have something very close to a WYSIWYG report at design time.

One row of data per page or one group per page

By default, all of the records are in one group, but you can set page level grouping to render one row of data on each page. This is ideal for something like a tax form that you want to print for every client or every employee, or an invoice that you want to print for every customer. For more information, see Grouping in a fixed page.

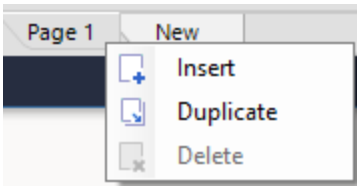
Where does the rest of the data go?

If there is data that does not fit within the space allocated for the data region at design time, you can assign it to flow into an OverflowPlaceholder control. This can go on the same page in a different area, for example, in the form of columns, or it can go on a separate page. For more information, see OverflowPlaceholder and Overflow Data in a Single Page.

Additional pages

You can run an entire report using the same page layout for every page, which is useful for something like an invoice, but does not satisfy every reporting need.

For other types of reports, you can add pages and create different layouts for each one, or duplicate a page you have already created. This can save a lot of time and effort when you have a report with many precisely placed controls, and you need additional pages that duplicate many of them. For example, when you need to provide employees with federal, state, and city copies of tax forms that have only one label changed. For more information, see Overflow Data in Multiple Pages.



You can also insert new pages between existing ones, and drag page tabs to rearrange them. With multiple pages, you can also choose how to collate the pages at run time. For more information, see how to Themes and Collate Multiple Copies of a Report.

> ⚠ **Caution**: Page Reports do not support nested data regions. A red border indicating overlapping of controls appears around the nested data region, on placing one data region inside another.

Report Definition Language (RDL) Report

The Report Definition Language (RDL) report is the most interactive type of report that we offer. Controls can grow and shrink, you can set up interactive sorting, you can set up drill-down reports in which detail data is initially hidden, and can be toggled by other items, and you can add drill-through links to other reports and to bookmark links within reports.

When you add a RDL report to a project, the OverflowPlaceholder control disappears from the toolbox, and the page tabs disappear from below the report design surface.

Master Reports

One way in which RDL reports differ from Page reports is the ability to create and use master reports. A master report is one that you use to add common report functionality like data, company logos, and page headers or footers, while using the ContentPlaceHolder control to designate areas where content reports can add data. In this way, you can quickly change the data source or company address and logo for an entire suite of reports in one convenient place. For more information, see Master Reports.

Page Break

RDL reports provides you the ability to add a page break by using the **PageSize** setting or by specifying the **PageBreakBefore** and **PageBreakAfter** properties of data region, group, and rectangle. In addition to the traditional print preview mode that allows you to consider the sheet size while printing a report, it also provides a galley mode wherein you can browse all your data in a single sheet. In RDL reports, you can easily verify data that does not include any page breaks, therefore it can be more appropriately used for browsing laterally extended reports that use controls like Tablix data region or for previewing reports that includes large amount of data.

Themes

Both Page and RDL reports can use themes to apply standard formatting to a range of report controls. Like using a master report, this allows you to change the look of a whole suite of reports in one place. You can specify colors for text and background, hyperlink colors, major and minor fonts, images, and constants, and then specify theme values in report control properties. When you want to change the look, you can do it all in the *.rdlx-theme file and it will apply to each report when it runs. For more information, see Create and Add Themes.

Data

RDL reports are ideal when you need to show data from different data sets, and when you do not need to control where the data appears on the page. Use data regions to display data in the report, and after the controls grow to accommodate your data, ActiveReports breaks it down into pages. For more information, see Data Sources and Datasets.

Shared Data Sources

RDL reports allow you to create and use shared data sources, so that you need not enter the same connection string every time you create a report.

Custom Resource Locators

You can create a custom resource locator for items to use in your reports. In this way, you can locate images for reports, or even reports to use in subreports or in drill-through links. For more information, see Custom Resource Locator.

Data Regions and Report Controls

All Rdl reports have controls that can display data differently than in section reports. You can use Sparkline and Bullet report controls for dashboard reports, plus there is a List, Table, and Tablix data regions to display your data. You can use

expressions in many of the properties to determine what to display and how to display it. For more information on these and other report controls, see Toolbox.

> **Note**: The PageHeader and PageFooter sections of RDL Reports do not display controls bound to dataset values at run time. These sections only display controls with static data like labels or, you can also add parameters using dataset values and use controls bound to parameter value in order to display dataset field values in these sections.

Data Visualizers

The Image and TextBox report controls have a Data Visualizer feature that allows you to display data in small, easy-to-comprehend graphs. This is a powerful tool to really make your data pop. For more information, see Data Visualizers.

Grouping

You can group data within data regions by fields or expressions, control the scope of aggregates, and even create recursive hierarchies in data with parent-child relationships. The Level function allows you to indent by level to show these relationships visually. For more information, see Grouping Data (Page Layout).

Interactivity

Interactive Sorting

You can allow users to sort data in List, BandedList, Table, or Tablix data regions using the Interactive Sort properties of a TextBox report control. For more information, see Allow Users to Sort Data in the Viewer.

Parameters

You can add parameters to reports that allow users to select which values to display in the report. These are also useful in creating drill-through reports. For more information, see Add Parameters.

Drill Down

You can use the ToggleItem property in the Visibility section for report controls, data regions, table rows, and tablix row and column groups to create drill-down reports. With these settings, you can initially hide items and set a toggle item that users can click to drill into more detailed data. For more information, see Create a Drill-Down Report.

Drill Through

You can use the Action property in the Navigation settings available on text boxes, images, and chart data values to create drill-through reports that let users click links to more detailed reports with parameters. Although you can create drill-through links to reports without parameters, this may leave users searching a huge detailed report for relevant information.

Bookmark Links

You can also use the Action property in the Navigation settings to jump to a bookmark or URL.

Pagination

You can control where pages break in RDL reports using PageSize settings, as well as PageBreakBefore and PageBreakAfter properties on data regions, groups, and rectangles.

Master Reports (RDL)

Master Reports are like dynamic templates you can design for use with content reports. This assists users in creating reports that share common elements such as a logo in the page header or a web site link in the page footer. You design the master report with controls, code, data sources, and layout properties that cannot be modified from content reports.

Master Reports differ from templates in that they are loaded each time the report is executed. Therefore, you can modify a master report and the changes to the master report automatically appear in any reports that reference it.

Designing Master Reports

When designing a master report, you use controls, code, data sources, and layout properties in the same way that you do in a normal report. A master report is valid on its own, and can be run without a content report. To prevent end users from modifying a master report, you can set permissions on the file to **Read Only** for that user or group.

In an RDL report, you can create a master report by saving it as an RDLX-master file. You can then apply it like a template to content reports.

A ContentPlaceHolder control appears in the toolbox when you convert an RDL report to a Master Report. This control defines regions where users can add content after applying a master report template.

> **Note:** In a section report (code-based report), there is a concept similar to Master Reports. However, here you create a base report class in a standard report that other reports inherit. See Inherit a Report Template for further information.

Creating Content Reports

The reports to which you apply the master report are content reports. A content report is not valid on its own, and cannot be run without its specified master report.

When the user creates a new report and sets a master report on it, the design view is effectively the opposite of the design view of the master report. Any report controls overlaid by ContentPlaceHolder controls are not visible in the content report at design time, but are visible at run time. These are the only areas where users can add report controls.

While designing the content report the user can

- Add elements that do not exist in the master report.
- Add new data sources that do not exist in the master report.
- Add new datasets from a data source in the master report.
- Add images to the EmbeddedImages collection.
- Add parameters to the ReportParameter collection.
- Add any number of report controls into placeholder rectangles designated by the master report.
- Modify the report name and description.
- Add new custom code that does not exist in the master report.

While designing the content report the user cannot

- Modify or remove elements that exist in the master report (disabled grey area).
- Remove a master report data source.
- Remove a master report dataset or modify its query.
- Modify the sort or filter on a master report dataset.
- Remove images from the EmbeddedImages collection.
- Remove parameters from the ReportParameter collection.
- Modify the margins or page settings of the master report.

> **Note:** Code in the master report is hidden in the content report, so in order to allow content report users to access code, the master report developer must provide information.

Run-Time Sequence of Events

This is what happens behind the scenes when you run a content report.

1. ActiveReports loads the content report.
2. The loader parses the master report tag on the content report and requests the master report from the resource resolver.
3. The master report is loaded into the definition.
4. As each ContentPlaceHolder in the content report is parsed, it finds the corresponding placeholder in the master report and loads the content from the content report into it.
5. Data sources, datasets, and fields are merged. The master report has higher priority if there is a conflict.
6. Themes are merged. The master report has higher priority if there is a conflict.
7. Report properties from the content report are added to those of the master report. For the following properties, the content report has a higher priority in case of conflict:
    - Report Description
    - Report Author
    - Report AutoRefresh
    - Report Custom
    - Report Language
    - Report DataTransform
    - Report DataSchema
    - Report ElementName
    - Report DataElementStyle
    - Dataset filters
    - Report Theme
    - Report Code
    - All content inside the ContentPlaceHolder controls

Modifying an Aggregated Report Definition

When you run a content report, the content report and its master combine to form an aggregated report definition. Using the ReportDefinition API, you can save this aggregate at run time as a third report definition which has no master or content report. Once this aggregate is saved as a normal report definition (*.rdlx file) you can edit it like any other report definition.

Advantages of a Master Report

- Implement common report functionality such as adding consistent page headers and footers within master reports.
- Apply company-wide changes in information such as address changes to a single master instead of modifying each report individually.
- Apply widespread data-related changes (such as data source location) to a single master report instead of modifying each report.
- Create code, data sources, themes and page layouts that are shared across the application or enterprise.
- Hide report complexity from end users who can use the stand-alone designer application to create content reports.

Advantages of a Shared Master Report

Shared master reports offer the advantages of a local master report, plus:

- They make your reports portable
- They allow multiple authors to use them

Code-Based Section Report

When you add an ActiveReports 14 Section Report (code-based) to your Visual Studio project, report layouts are saved as C# or Visual Basic files within the project in which they are created. These files are compiled into the application when you build it. Each report is composed of three files:

- *rptYourReportName*.vb or .cs
- *rptYourReportName*.Designer.vb or .cs
- *rptYourReportName*.resx

In this way, layout information models the behavior of Windows Forms in the .NET framework.

The design surface of a section report has banded sections that repeat depending on the data and the type of section. For more information, see Section Report Structure and Section Report Events.

Code

This type of report is the most flexible in terms of what a .NET developer can achieve using code. It has an extensive API and is event-based, which allows you to control all aspects of the report and how it is generated. If you like, you can even build a report completely in code. See details about the API in the **Class Library (on-line documentation)** section of the help.

The API is also available with XML-based section reports, but you use VB or C# script instead of Windows Forms-like code. For more information, see XML-Based Section Report .

Data

Code-based section reports connect to data either via settings that you specify in the Report Data Source dialog, or through code. You can find more information on all of the ways to connect to data in a section report in the Report Data topic.

Viewing and Exporting

To display a code-based report in the viewer, you use the LoadDocument method of the viewer. See Preview Reports for more information. To export a code-based report, you use the Export method of the export you choose.

XML-Based Section Report

When you add an ActiveReports 14 Section Report (xml-based) report to your Visual Studio project, the layout is saved as a stand-alone Report XML (RPX) file. Since these files are not compiled into your application, they are a good option for solutions in which you need to update or add reports frequently.

The RPX format cannot contain Visual Basic.NET or C# code. Instead, you can add VB.NET or C# script in the Script view of the report.

For more information on using script with a layout file, see Scripting in Section Reports.

XML-based section reports are the same as Code-Based Section Report with regard to data, events, structure, and exports, but everything is contained in a single, portable RPX file.

End User Report Designer

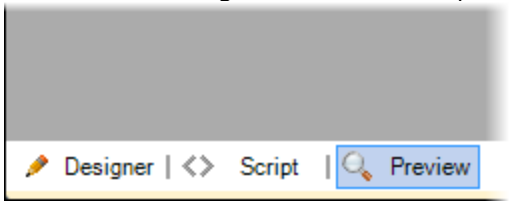
If you want to allow end users to edit and create section reports in a Windows Forms application you create with the Designer control, these are XML-based, as there is nowhere to put Visual Studio code and no way to handle multiple files for a code-based section report. For more information, see Creating a Basic End User Report Designer (Pro Edition).

Preview Reports

ActiveReports provides a number of ways to view your report output. You have an option of previewing the report as you create it in a Visual Studio project at design time.

Previewing Reports at Design Time

ActiveReports makes it easy for you to preview your report while you are still creating it. Click the Preview tab at the bottom of the designer and see the output as it appears in a viewer. See Designer Tabs for further information.



With the in-built Viewers for Windows Forms and Web, you can view your report in any of these platforms as well in a separate viewer control. This section introduces all the available report viewing options.

Topic	Content
Windows Forms	This section explains how to view a report in the Windows Forms Viewer and demonstrates the Viewer's features, touch gestures and shortcut keys.

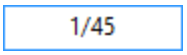
ASP.NET	This section introduces the WebViewer where you can view your report output in various types of viewers and provides key features of each viewer type.
JavaScript Viewer	This section describes the JSViewer and how to configure a target ASP.NET backend application with the JSViewer.
WPF	This section describes the WPF Viewer toolbar, its additional features and how to view a report in the WPF viewer.
Medium Trust Support	Learn about the features and limitations available in Medium Trust Support environment.
Viewing Reports from Different Domains using CORS	Learn about accessing the reports from different domains using Cross-Origin Resource Sharing technology.

Windows Forms Viewer

Besides previewing your report at design time, you can also view the reports you design in the Viewer. This viewer contains a toolbar and a sidebar with **Thumbnails**, **Search results**, **Document map** and **Parameters** panes.

Viewer Toolbar

The following table lists the actions you can perform through the Viewer toolbar.

Toolbar Element	Name	Description
	First page	Takes you to the first page of the report. This button is enabled when a page other than the first page is open.
	Last page	Takes you to the last page of the report. This button is disabled on reaching the last page of the report.
	Previous page	Takes you to the page prior to the current page. This button is enabled when a page other than the first page is open.
	Next page	Takes you to the page following the current page. This button is disabled on reaching the last page of the report.
	Current page	Opens a specific page in the report. To view a specific page, type the page number and press the Enter key.
	Backward	Takes you to the last viewed page. This button is enabled when you move to any page from the initial report page. Clicking this button for the first time also enables the **Forward** button.

	Forward	Takes you to last viewed page before you clicked the Backward button. This button is enabled once you click the **Backward** button.
	Back to parent report	Returns you to the parent report in a drillthrough report.
	Default	Allows you to specify a default mouse pointer mode.
	Pan mode	A hand symbol serves as the cursor that you can use to navigate.
	Selection mode	Allows you to select contents on the report. Click the Copy icon (see image and description below) to copy the selected content to the clipboard.
	Snapshot mode	Allows you to select content on the report that you can paste as an image into any application that accepts pasted images.
	Toggle sidebar	Displays the sidebar that includes the Thumbnails, Parameters, Document map and Search results panes.
	Print	Displays the Print dialog where you can specify the printing options.
	Galley mode	Provides a viewer mode which removes automatic page breaks from a Report Definition Language (RDL) and displays data in a single scrollable page. This mode maintains page breaks you create in the report and removes only automatic page breaks.
	Copy	Copies text that you select in the Selection mode to the clipboard. **Note:** In case the GrapeCity.ActiveReports.Export.Xml.dll and GrapeCity.ActiveReports.Export.Word.dll are not available in GAC, you might need to add references to these assembly files to enable the viewer's Copy button.
	Find	Displays the Find dialog to find any text in the report.
	Zoom out	Decreases the magnification of your report.
	Current zoom	Displays the current zoom percentage which can also be edited.

	Zoom in	Increases the magnification of your report.
	Fit width	Fits the width of the page according to viewer dimensions.
	Fit page	Fits the whole page within the current viewer dimensions.
	Single page view	Shows one page at a time in the viewer.
	Continuous view	Shows all preview pages one below the other.
	Multipage view	Offers you an option to select how many pages to preview in the viewer at one time.
	Refresh	Refreshes the report. ⚠ **Caution**: Refresh button gets disabled when you load a section report in the Viewer control through any of the following: • **Document Property (on-line documentation)** • **LoadDocument(SectionDocument) Method ('LoadDocument Method' in the on-line documentation)** • **LoadDocument(String) Method ('LoadDocument Method' in the on-line documentation)**
	Cancel	Cancels the report rendering.
	Touch Mode	Allows you to select touch mode for the Viewer. **Note:** The **Touch Mode** button only appears on the toolbar while working on touch enabled devices.

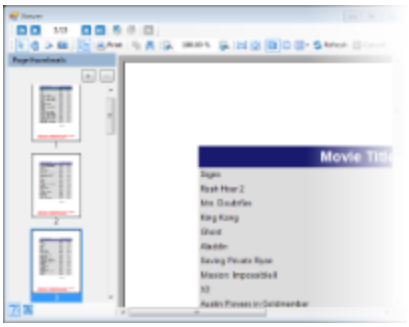
Viewer Sidebar

The Viewer sidebar appears on the left of the Viewer control when you click the **Toggle sidebar** button in the toolbar. By default, this sidebar shows the Thumbnails and Search Results panes. The additional Document map and Parameters also appear in this sidebar. You can toggle between any of the viewer panes by clicking the buttons for each pane at the bottom of the sidebar.

Thumbnails pane

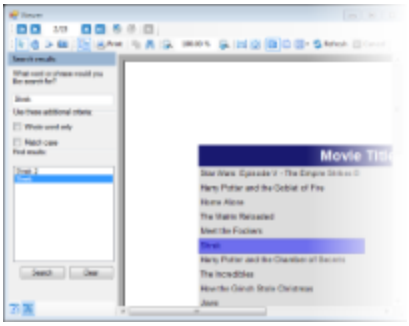
The **Thumbnails** pane appears by default in the sidebar when you click the **Toggle sidebar** button in the toolbar.

This pane is composed of a thumbnail view of all the pages in a report. Click any thumbnail to navigate directly to the selected report page. You can also modify the size of the thumbnail when you click (+) or (-) button to zoom in and zoom out.



Search results pane

The **Search** pane is the other default pane besides Thumbnails that appears in the sidebar when you click the **Toggle sidebar** button. This pane lets you enter a word or phrase from which to search within the report.



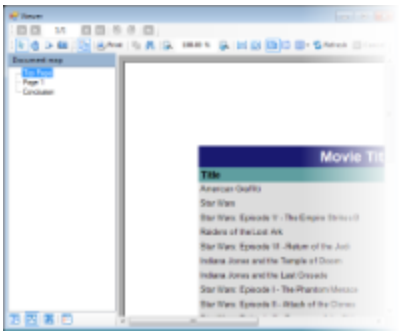
To search in a report:

- Enter the word or phrase in the search field.
- Under **Use these additional criteria**, you may optionally choose to search for the whole word or match the case of the search string while searching in the report.
- Click the **Search** button to see the results appear in the **Find results** list.
- Click an item in the list to jump to that item in the report and highlight it.

To start a new search or clear the current search results, click the **Clear** button under the **Find results** list.

Document map pane

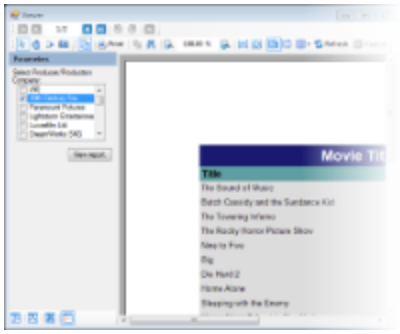
The Documents map pane is enabled for reports where the Label property or the Document map label is set. This pane displays each value for the text box, group, or sub report that you label, and you can click them to navigate to the corresponding area of the report in the Viewer.



If a report does not have the Label property or Document map label set, the Documents map pane does not appear in the sidebar.

Parameters pane

The Viewer allows you to view reports with parameters. In the toolbar, click the **Toggle sidebar** button to open the Viewer sidebar and if your report contains parameters, the **Parameters** pane shows up automatically.



1. In the **Parameters** pane, you are prompted to enter a value by which to filter the data to display.
2. Enter a value or set of values and click **View report**, to filter the report data and display the report.

If a report does not have parameters, the Parameters pane does not appear in the sidebar.

> **Note:** ActiveReports stopped using the ClearType technology in rendering text in a font system by default.

Display Report Output in the Viewer

The following code examples demonstrate how you can display the report output in the Viewer.

1. In a Visual Studio Windows Forms application, from the Visual Studio toolbox, drag the Viewer control onto your Windows Form.
2. Set the viewer's **Dock** property to **Fill** to show the complete Viewer control on the Form.
3. Double-click the title bar of the Form to create an event-handling method for the **Form_Load** event.
4. In the Form_Load event, add code like the following to run the report and display it in the viewer. Each of these code snippets presumes a report in the project of the type indicated with the default name. (If you have renamed your report, you need to rename it in the code as well)

   To write the code in Visual Basic.NET

   The following example demonstrates how you display a page report in the Viewer control.

   Visual Basic. NET code. Paste INSIDE the Form_Load event.

   ```
   Dim file_name As String = "..\..\PageReport1.rdlx"
   Dim pageReport As New GrapeCity.ActiveReports.PageReport(New
   System.IO.FileInfo(file_name))
   Dim pageDocument As New GrapeCity.ActiveReports.Document.PageDocument(pageReport)
   Viewer1.LoadDocument(pageDocument)
   ```

   The following example demonstrates how you display a RDL Report in the Viewer control.

Visual Basic. NET code. Paste INSIDE the Form_Load event.

```
Dim file_name As String = "..\..\RdlReport1.rdlx"
Dim pageReport As New GrapeCity.ActiveReports.PageReport(New
System.IO.FileInfo(file_name))
Dim pageDocument As New GrapeCity.ActiveReports.Document.PageDocument(pageReport)
Viewer1.LoadDocument(pageDocument)
```

The following example demonstrates how you can display a section report (code-based) in the Viewer control.

Visual Basic. NET code. Paste INSIDE the Form_Load event.

```
Dim sectionReport As New SectionReport1()
Viewer1.LoadDocument(sectionReport)
```

The following example demonstrates how you can display a section report (xml-based) in the Viewer control.

Visual Basic. NET code. Paste INSIDE the Form_Load event.

```
Dim sectionReport As New GrapeCity.ActiveReports.SectionReport()
Dim xtr As New System.Xml.XmlTextReader("..\..\SectionReport1.rpx")
sectionReport.LoadLayout(xtr)
xtr.Close()
Viewer1.LoadDocument(sectionReport)
```

To write the code in C#

The following example demonstrates how you display a page report in the Viewer control.

C# code. Paste INSIDE the Form_Load event.

```
string file_name = @"..\..\PageReport1.rdlx";
GrapeCity.ActiveReports.PageReport pageReport = new
GrapeCity.ActiveReports.PageReport(new System.IO.FileInfo(file_name));
GrapeCity.ActiveReports.Document.PageDocument pageDocument = new
GrapeCity.ActiveReports.Document.PageDocument(pageReport);
viewer1.LoadDocument(pageDocument);
```

The following example demonstrates how you display a RDL Report in the Viewer control.

C# code. Paste INSIDE the Form_Load event.

```
string file_name = @"..\..\RdlReport1.rdlx";
GrapeCity.ActiveReports.PageReport pageReport = new
GrapeCity.ActiveReports.PageReport(new System.IO.FileInfo(file_name));
GrapeCity.ActiveReports.Document.PageDocument pageDocument = new
GrapeCity.ActiveReports.Document.PageDocument(pageReport);
viewer1.LoadDocument(pageDocument);
```

The following example demonstrates how you can display a section report (code-based) in the Viewer control.

C# code. Paste INSIDE the Form_Load event.

```
SectionReport1 sectionReport = new SectionReport1();
viewer1.LoadDocument(sectionReport);
```

The following example demonstrates how you can display a section report (xml-based) in the Viewer control.

C# code. Paste INSIDE the Form_Load event

```
GrapeCity.ActiveReports.SectionReport sectionReport = new
GrapeCity.ActiveReports.SectionReport();
System.Xml.XmlTextReader xtr = new
System.Xml.XmlTextReader(@"..\..\SectionReport1.rpx");
sectionReport.LoadLayout(xtr);
xtr.Close();
viewer1.LoadDocument(sectionReport);
```

Additional Features

Following is an introduction to the additional capabilities of the Viewer to guide you on using it effectively:

Split windows

1. Run your viewer project.
2. Click above the vertical scrollbar to grab the splitter control and drag downward.
3. With the viewer split into two sections, you can easily compare report pages.

Advanced Printing Options

Viewer provides advanced printing options that allow you to control the report page layout and watermark settings through the Page Setup dialog. In this dialog, you can also preview the report as it would appear with each print setting. See Advanced Print Options for further details.

You can also set the **PrintingSettings ('PrintingSettings Property' in the on-line documentation)** property of the Viewer to directly print without displaying a dialog or to switch from a ActiveReports print dialog to a .NET Framework standard print dialog (System.Windows.Forms.PrintDialog) on clicking the **Print** button on the Viewer toolbar. You can set the **PrintingSettings** property of the Viewer control from the Properties window.

PrintingSettings property provides the following options:

PrintingSettings Options	Description
ShowPrintDialog	Displays a dialog in which the user can set the printer options before printing.
ShowPrintProgressDialog	Displays a print progress dialog, in which the user can cancel the printing job.
UsePrintingThread	Specifies whether printing should be performed for individual threads or not.
UseStandardDialog	Specifies whether to use the .NET Framework standard print dialog (System.Windows.Forms.PrintDialog) while printing the document (section or page).

Exporting

Use the Export Filters to export a page or a section report to different formats directly from the Viewer. After you load the document in the Viewer, you can use a sample code like the following which shows one overload of the **Export ('Export Method' in the on-line documentation)** method with a PDF export filter. This code creates an outputPDF.pdf file in the bin\debug folder of your project.

To write the code in Visual Basic.NET

Visual Basic. NET code. Paste INSIDE an event like Button_Click event.

```
Dim PDFEx As New GrapeCity.ActiveReports.Export.Pdf.Section.PdfExport
Viewer1.Export(PDFEx, New FileInfo(Application.StartupPath + "\outputPDF.pdf"))
```

To write the code in C#

C# code. Paste INSIDE an event like Button_Click event.

```
GrapeCity.ActiveReports.Export.Pdf.Section.PdfExport PDFEx = new
GrapeCity.ActiveReports.Export.Pdf.Section.PdfExport();
viewer1.Export(PDFEx, new System.IO.FileInfo (Application.StartupPath +
"\\outputPDF.pdf" ));
```

Note: Make sure that you add a reference to the required export assembly in your project before setting the export filter in code. See Export Filters further details.

Annotations Toolbar

You can use annotations when working with a report in the Viewer and add notes, special instructions or images directly to the reports.

Annotations are added via the Viewer's toolbar, which is hidden by default. You can make the Annotations toolbar available by setting the **AnnotationDropDownVisible** property to true in the viewer's properties grid.

Annotation Name	Description
AnnotationText	A rectangular box in which you can enter text.
AnnotationCircle	A circle without text. You can change the shape to an oval.
AnnotationRectangle	A rectangular box without text.
AnnotationArrow	A 2D arrow in which you can enter text. You can change the arrow direction.
AnnotationBalloon	A balloon caption in which you can enter text. You can point the balloon's tail in any direction.
AnnotationLine	A line with text above or below it. You can add arrow caps to one or both ends and select different dash styles.
AnnotationImage	A rectangle with a background image and text. You can select an image and its position, and place text on the image.

Keyboard Shortcuts

The following shortcuts are available on the Viewer:

Keyboard Shortcut	Action
Ctrl + F	Shows the find dialog.
Ctrl + P	Shows the print dialog.
Esc	Closes the find or print dialogs.
Page Down	Moves to the next page.
Page Up	Moves to the previous page.
Ctrl + T	Shows or hides the table of contents.
Ctrl + Home	Moves to the first page.
Ctrl + End	Moves to the last page.
Ctrl + Right	Navigates forward.
Ctrl + Left	Navigates backward.
Ctrl + -	Zooms out.
Ctrl + +	Zooms in.
Left, Right, Up, Down	Moves the visible area of the page in the corresponding direction.
Ctrl + 0 (zero)	Sets the zoom level to 100%.
Ctrl + rotate mouse wheel	Changes the zoom level up or down.
Ctrl + M	Turns on the continuous view.
Ctrl + S	Turns off the continuous view.
Ctrl + I	Shows multiple pages.
Ctrl + G	Focuses on PageNumber area and selects content.
F5	Refreshes the report.
Home	Moves to the start of the current page.
End	Moves to the end of the current page.

Viewer's Thumbnails pane shortcut keys

You can use the following shortcut keys while using the thumbnails pane in the Viewer.

Keyboard Shortcut	Action
Up Arrow	Goes to the previous page.
Down Arrow	Goes to the next page.

Right Arrow	Goes to right page. If no thumbnail exist on the right, it goes to the next page.
Left Arrow	Goes to left page. If no thumbnail exist on the left, it goes to the previous page.
Page Down	Scroll to the next thumbnail's view port. It also keep the current selected page unchanged.
Page Up	Scroll to the previous thumbnail's view port. It also keep the current selected page unchanged.
Home	Goes to the first page.
End	Goes to last page.

Touch Support

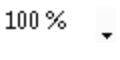
ActiveReports introduces touch support for **Windows Viewer**. This feature gives you the flexibility to interact with the Viewer using simple touch gestures. Now you can install ActiveReports on any touch enabled Windows device and view the reports anywhere you are.

You can switch to the touch mode by just clicking the Touch mode button on the Viewer toolbar.

> **Note:** In touch mode, you can still use the mouse to perform ActiveReports operations.

Touch mode Toolbar

The following table lists the actions you can perform through the Viewer toolbar.

Icon	Function	Details
	Sidebar	Displays the sidebar that includes the Thumbnails, Parameters, Document map and Search results panes.
	Print	Displays the Print dialog where you can specify the printing options.
	Galley mode	Provides a viewer mode which removes automatic page breaks from a Report Definition Language (RDL) and displays data in a single page. This mode maintains page breaks you create in the report and removes only automatic page breaks. For RDL report only.
	Copy	Copies text that you select in the Selection mode to the clipboard. **Note:** In case the GrapeCity.ActiveReports.Export.Xml.dll and GrapeCity.ActiveReports.Export.Word.dll are not available in GAC, you might need to add references to these assembly files to enable the viewer's Copy button.
	Find	Displays the Find dialog to find any text in the report.
	Current zoom	Displays the current zoom percentage which can also be edited.

	Single page view	Shows one page at a time in the viewer.
	Continuous page	Shows all preview pages one below the other.
	Multiple page	Offers you an option to select how many pages to preview in the viewer at one time.
	Previous page	Takes you to the page prior to the current page. This button is enabled when a page other than the first page is open.
1/8	Current page	Opens a specific page in the report. To view a specific page, type the page number and press the Enter key.
	Next page	Takes you to the page following the current page. This button is disabled on reaching the last page of the report.
	Back to parent report	Returns you to the parent report in a drillthrough report.
	Refresh	Refreshes the Viewer. ⚠ **Caution**: Refresh button gets disabled when you load a section report in the Viewer control through any of the following: • **Document Property (on-line documentation)** • **LoadDocument(SectionDocument) Method ('LoadDocument Method' in the on-line documentation)** • **LoadDocument(String) Method ('LoadDocument Method' in the on-line documentation)**
	Cancel	Cancels the report rendering.
	Touch mode	Allows you to select touch mode for the Viewer.

Context menu

To display the context menu, you must tap and hold in the preview area.

Icon	Function	Gesture	Details
	Pan mode	Tap	Hand cursor used to move the visible portion of the reports. Drag the hand tool in the direction you want to move the report.
	Selection mode	Tap	Allows you to select contents on the report. Click the Copy icon (see image and description above) to copy the selected content to the clipboard.
	Snapshot mode	Tap	Allows you to select content on the report that you can paste as an image into any application that accepts pasted images.

Preview

Gesture	Gesture (Image)	Preview Area Element	Details
Flick		Single Page View	Flick in the vertical or horizontal direction moves to the next or previous page. A single flick moves single page.
		Continuous Page View	Performs the page scrolling.
Tap		Link	Opens links (URL, drill-though links, etc).
Pan		Slider	Moves the slider.
		Border between Sidebar and Preview Area	Moves the border. **Note:** The pan gesture does not work in the report area where **ScrollbarEnabled** property is set to **False**.
		Selection mode, Snapshot mode	Pan from selection start point to selection end point.
Pinch		Preview Area	Pinch to zoom out.
Stretch		Preview Area	Stretch to zoom in.
Double tap		Preview Area	Tap twice to change the view mode (from the Single page view to the Multiple page mode).
Tap and hold		Selection mode, Snapshot mode	Displays the context menu where you can select the pan, selection, or snapshot mode.

Customize the Viewer ToolStrip

There are a number of ways in which you can customize the Viewer control to make it a perfect fit for your Windows application. You can add and remove buttons from the toolbars, add and remove menu items, create custom dialogs, and call them from custom click events.

You can use the methods listed in the System.Windows.Forms.ToolStripItemCollection documentation on MSDN to customize each ToolStrip.

ToolStrip

TheToolStrip contains the following ToolStripItems by index number.

- 0 Toggle sidebar

- 1 Separator
- 2 Print
- 3 Galley mode
- 4 Separator
- 5 Copy
- 6 Find
- 7 Separator
- 8 Zoom out
- 9 Zoom In
- 10 Current Zoom
- 11 Separator
- 12 Fit width
- 13 Fit page
- 14 Separator
- 15 Single page
- 16 Continuous mode
- 17 Multipage mode
- 18 Separator
- 19 First page
- 20 Previous page
- 21 Current
- 22 Next page
- 23 Last page
- 24 Separator
- 25 History back
- 26 History forward
- 27 Separator
- 28 Back to parent
- 29 Separator
- 30 Refresh
- 31 Cancel button
- 32 Separator
- 33 Pan mode
- 34 Copy select
- 35 Snapshot
- 36 Separator
- 37 Annotations

You can access these ToolStripItems by index with the **Insert** and **RemoveAt** methods. Other methods, including **Add** and **AddRange**, are described in the System.Windows.Forms.ToolStripItemCollection documentation on MSDN.

ToolStrip Implementation

When you add a new item to a ToolStrip, you need to add an ItemClicked event handler and an ItemClicked event for the ToolStrip with the new item. At run time, when a user clicks the new ToolStrip item, they raise the ItemClicked event of the ToolStrip containing the item

Add the event handler to the **Load** event of the Form that contains the Viewer control, and use the IntelliSense Generate Method Stub feature to create the related event. For examples of the code to create an event hander, see the Customize the Viewer Control topic, and the Custom Preview sample.

Customize the Viewer Control

ActiveReports includes a Viewer control for Windows Forms that lets you show report output in a custom preview form. You can modify both viewer's mouse mode and touch mode toolbars and set custom commands. For example, in the following sample codes we show customization specific to mouse mode toolbar and touch mode toolbar.

To create a basic preview form

1. In Visual Studio, create a new Windows Forms project.
2. From the Visual Studio toolbox on the ActiveReports 14 tab, drag the **Viewer** control onto the form. If you do not have the Viewer in your toolbox, see Quick Start.
3. With the viewer control selected, in the Properties window, set the **Dock** property to **Fill**.
4. From the **Project** menu, select **Add New Item**.
5. Select **ActiveReports 14 Section Report (code-based)** and click the **Add** button.
6. Double-click in the title bar of the form to create a Form Load event.
7. Add the following code to run the report and display the resulting document in the viewer.

   To write the code in Visual Basic.NET

   Visual Basic.NET code. Paste INSIDE the Form Load event.

   ```
   Dim rpt as new SectionReport1
   Viewer1.LoadDocument(rpt)
   ```

   To write the code in C#

   C# code. Paste INSIDE the Form Load event.

   ```
   SectionReport1 rpt = new SectionReport1();
   viewer1.LoadDocument(rpt);
   ```

8. Press **F5** to run the project.

Customize the Mouse Mode Toolbar

1. Add a second Windows Form to the project created above and name it **frmPrintDlg**.
2. Add a label to **frmPrintDlg** and change the **Text** property to **This is the custom print dialog**.
3. Add a button to **frmPrintDlg** and change the **Text** property to **OK**.
4. Back on the viewer form, double-click the title bar of the form to go to the Form Load event.
5. Add the following code to the Form Load event to remove the default print button and add your own.

   To write the code in Visual Basic.NET

   Visual Basic.NET code. Paste INSIDE the Form Load event.

   ```
   'Remove the print button.
   Viewer1.Toolbar.ToolStrip.Items.RemoveAt(2)
   'Remove the extra separator.
   Viewer1.Toolbar.ToolStrip.Items.RemoveAt(1)
   'Add a new button to the end of the tool strip with the caption "Print."
   Dim tsbPrint As New ToolStripButton("Print")
   Viewer1.Toolbar.ToolStrip.Items.Add(tsbPrint)
   'Create a click event handler for the button.
   ```

```
AddHandler tsbPrint.Click, AddressOf tsbPrint_Click
```

To write the code in C#

C# code. Paste INSIDE the Form Load event.

```
//Remove the print button.
viewer1.Toolbar.ToolStrip.Items.RemoveAt(2);
//Remove the extra separator.
viewer1.Toolbar.ToolStrip.Items.RemoveAt(1);
//Add a new button to the end of the tool strip with the caption "Print."
ToolStripButton tsbPrint = new ToolStripButton("Print");
viewer1.Toolbar.ToolStrip.Items.Add(tsbPrint);
//Create a click event handler for the button.
tsbPrint.Click += new EventHandler(tsbPrint_Click);
```

6. Add the following code to the Form class below the Load event to display frmPrintDlg when a user clicks the custom print button.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste BELOW the Form Load event.

```
'Call the custom dialog from the new button's click event.
Private Sub tsbPrint_Click(sender As Object, e As EventArgs)
    Me.CustomPrint()
End Sub

'Call the custom print dialog.
Private Sub CustomPrint()
    Dim _printForm As New frmPrintDlg()
    _printForm.ShowDialog(Me)
End Sub
```

To write the code in C#

C# code. Paste BELOW the Form Load event.

```
//Call the custom dialog from the new button's click event.
private void tsbPrint_Click(object sender, EventArgs e)
{
    this.CustomPrint();
}

//Call the custom print dialog.
private void CustomPrint()
{
    frmPrintDlg _printForm = new frmPrintDlg();
    _printForm.ShowDialog(this);
}
```

7. Press **F5** to run the project and click on the print button of main viewer to view custom print dialog.

Customize the Touch Mode Toolbar

1. Double-click in the title bar of the Viewer form to create a Form Load event.
2. Add the following code to add a custom Zoom out button.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the Form Load event.

```
Dim zoomOutButton = viewer1.TouchModeToolbar.ToolStrip.Items(8)
zoomOutButton.Visible = true
```

To write the code in C#

C# code. Paste INSIDE the Form Load event.

```
var zoomOutButton = viewer1.TouchModeToolbar.ToolStrip.Items[8];
zoomOutButton.Visible = true;
```

> ⚠ **Caution:** Any customization done in mouse mode does not apply to the touch mode and vice versa.

Although this topic and the sample both demonstrate using section reports, customized viewers support page reports and RDL reports as well. The only difference is in the way that you load reports. For more information, see Windows Forms Viewer.

WebViewer (ASP.NET)

Professional Edition

With the Professional Edition license, you can use the WebViewer control to quickly display reports in any of the three viewer types: **HtmlViewer**, **RawHtml**, or **AcrobatReader**.

> ⚠ **Important**:
>
> - Before using the WebViewer control, you must first Configure HTTPHandlers in IIS 8 and IIS 10.

Standard Edition

With the Standard Edition license, you can export reports to use on the Web or use Web Services to distribute reports or data sources. For more information on Web exporting, please see the Custom Web Exporting (Std Edition) section.

In this section

Topic	Content
Getting Started with the WebViewer	Explore the ways that the WebViewer control can save you time.

Using the HTML Viewer	Learn about the features available with the HTML viewer, including parameters, table of contents, search, and the toolbar.
Using Javascript with the HTML Viewer	Learn about how you can use HTML Viewer using Javascript.

Getting Started with the WebViewer

The WebViewer control that is licensed with the Professional Edition allows you to quickly display reports in Web applications. Once you drop the control onto a Web Form, you can look in the Visual Studio Properties grid and select the **ViewerType ('ViewerType Property' in the on-line documentation)** that you want to use.

The WebViewer control supports the following types:

- **HTMLViewer** (default): Provides a scrollable view of a single page of the report at a time. Downloads only HTML and javascript to the client browser. Not recommended for printable output. See the Using the HTML Viewer topic for details.
- **RawHTML**: Shows all pages in the report document as one continuous HTML page. Provides a static view of the entire report document, and generally printable output, although under some circumstances pagination is not preserved.
- **AcrobatReader**: Returns output as a PDF document viewable in Acrobat Reader.
  Client requirements: Adobe Acrobat Reader

In a WebViewer, an RDL report can be rendered in two modes - Paginated and Galley. Using galley mode, you can view the contents of the RDL report in a single and scrollable page. You can set Galley mode through UI of the WebViewer or through code by setting **RenderMode** property to **Galley ('RenderMode Enumeration' in the on-line documentation)**.

> **Note**: The WebViewer and JSViewer are supported only in the **Integrated pipeline mode**. You will get PlatformNotSupportedException on using these Viewers in Classic pipeline mode.

To use the WebViewer control

1. In VSIDE, create a new ASP.NET Web Forms Application.
2. To install nuget pakage for **GrapeCity.ActiveReports.Web**, go to **Tools > Nuget Package Manager > Manage Nuget Packages for Solution...**, browse for the package and click **Install**.
3. In Solution Explorer, right-click the project and select **Add > New Item**.
4. Select WebForm and click **Add**.
5. Go to the **Design** tab of the newly added WebForm and drag and drop the WebViewer control to the WebForm designer.

   > **Note**: if you get an error on adding the WebViewer control, you should install or upgrade the Microsoft.CodeDom.Providers.DotNetCompilerPlatform NuGet package. See Troubleshooting for details.

To preview Code-Based Section Reports in WebViewer control

You need to update the **Global.asax** file as follows:

Global.asax.cs

```
public class Global : System.Web.HttpApplication
    {
        protected void Application_Start(object sender, EventArgs e)
        {
            this.UseReporting(settings =>
            {
                settings.UseFileStore(new DirectoryInfo(Server.MapPath("~")));
                settings.UseCompression = true;
                settings.UseCustomStore(GetReport);
            });
        }
        public object GetReport(string reportName = "SectionReport")
        {
            SectionReport1 rpt = new SectionReport1();
            return rpt;
        }
    }
```

Global.asax.vb

```
Public Class _Global
    Inherits System.Web.HttpApplication
    Protected Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
        Me.UseReporting(Sub(settings)
                            settings.UseFileStore(New
DirectoryInfo(Server.MapPath("~")))
                            settings.UseCompression = True
                            settings.UseCustomStore(AddressOf GetReport)
                        End Sub)
    End Sub
    Public Function GetReport(ByVal Optional reportName As String = "SectionReport") As
Object
        Dim rpt As SectionReport1 = New SectionReport1()
        Return rpt
    End Function
End Class
```

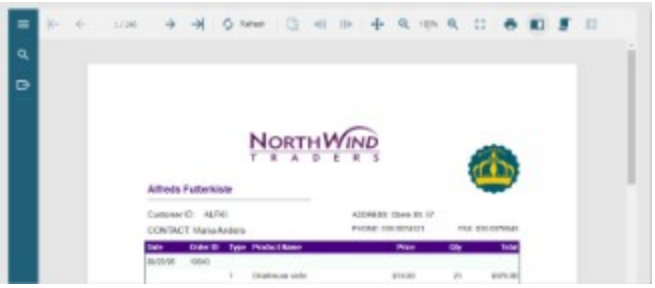
Using the HTML Viewer

HTML Viewer is the default viewer type of the WebViewer control, and provides a scrollable view of the report one page at a time. It includes HTML representations of the toolbar as well as the sidebar that contains **Parameters**, **Export** and **Search** panes.

HTML Viewer Properties

Property	Description
BookmarkStyle	Specify whether to use HTML bookmarks, or none.
CharacterSet	Select from 15 character sets to use for the report.
IncludePageMargins	Specify whether to keep page margins on reports in the generated HTML.
OutputType	Specify whether to use DHTML or HTML for the output.
RemoveVerticalSpace	Specify whether to keep white space, for example at the end of a page not filled with data before a page break.

The HTML Viewer downloads only HTML and javascript to the client browser.



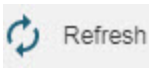
HTML Viewer Sidebar

Sidebar Element	Name	Description
	Display Sidebar	Displays the sidebar that includes the Search, Export, and Parameters panes.
	Search	Displays the Search pane.
	Export	Displays the Export pane where you can select an export format and options for the report you are previewing.
	Parameters	Displays the Parameters pane. If a report does not have parameters, the Parameters button not displayed.

HTML Viewer Toolbar

The HTML viewer toolbar offers various ways to navigate through reports.

Toolbar Element	Name	Description
	Go to First/Last page	Jumps to the first or last page of a report.

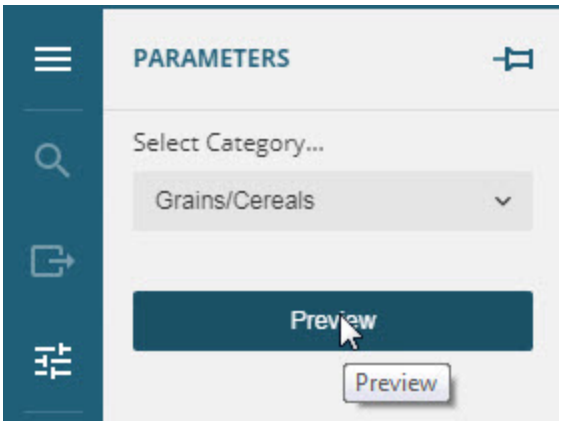
	Go to Previous/Next page	Navigates through a report page by page.
	Current page	Displays the current page number and page total. Enter the page number to view a specific page.
	Refresh	Refreshes the report.
	History: Back to Parent	Returns to the parent report in a drill-down page report or RDL report.
	History: Go Back	Navigates to a previous page in a parent report in a drill-down page report or RDL report.
	History: Go Forward	Navigates to a next page in a parent report in a drill-down page report or RDL report.
	Move Tool	A move tool that you can use to navigate the report.
	Zoom Out	Decreases the magnification of your report.
	Zoom mode	Displays the current zoom percentage which can also be edited. Allows you to select from the available zoom options - **50%**, **100%**, **150%**, **200%, 300%**, **Fit to Page**, and **Fit to Width**.
	Zoom In	Increases the magnification of your report.
	Toggle Fullscreen	Switches to a Fullsreen mode.
	Print	Displays the Print screen to specify printing options.
	Single Page View	Shows one page at a time in the viewer.
	Continious View	Shows all preview pages one below the other.
	Galley mode	Provides a viewer mode which removes automatic page breaks from an RDL report and displays data in a single scrollable page. This mode maintains page breaks you create in the report.

HTML Viewer Parameters

The HTML viewer allows you to view reports with parameters. The Parameters pane shows up automatically. To show or hide the Parameters pane in the sidebar, click the **Toggle Sidebar** button in the Toolbar.

In the **Parameters** pane, you are asked to enter a value by which to filter the data to display.

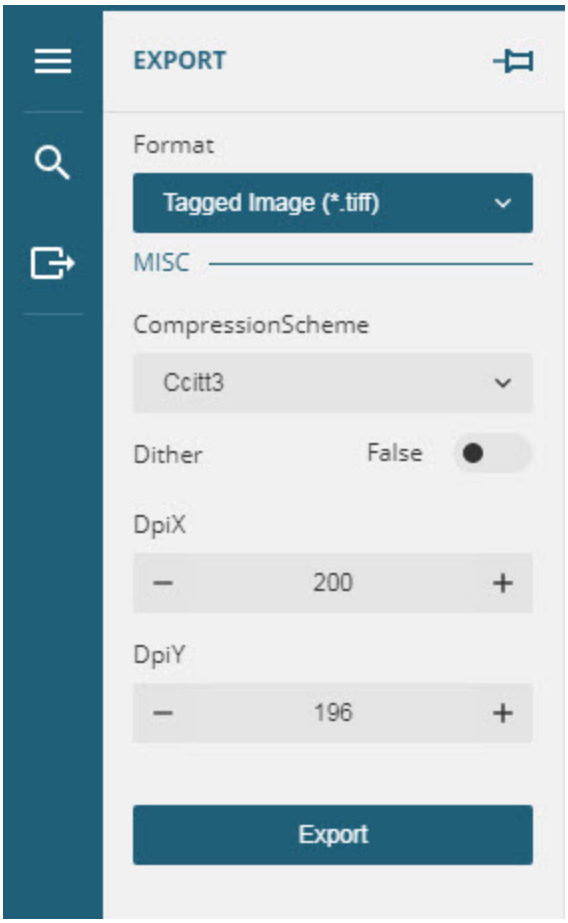
To filter the report data, enter a value or set of values and click **Preview**.



If a report does not have parameters, the Parameters pane of the sidebar is not displayed.

HTML Viewer Export

To display the Export pane, click **Export** in the sidebar. The **Export** pane lets you enter parameters for exporting a report that you are previewing. The available format options are Excel 2003, Excel, Word 2003, Word, PDF, CSV, JSON, XML, Tagged Image, and Web Archive.



After you set all necessary export properties, click **Export**.

HTML Viewer Search

The **Search** pane lets you enter a word or phrase for which to search within the report. Under **Use these additional criteria**, you may optionally select additional criteria. When you click **Search**, any results appear in the **Find results** list. Click an item in the list to jump to the item you selected and highlight it.

Using Javascript with the HTML Viewer

To use JavaScript with the WebViewer, initialize the WebViewer's view model using the clientId returned from the WebViewer. Once you initialize the view model, you can access its API methods and properties to modify the WebViewer. You can also use the same view model to access the API for the side bar search panel and the toolbar.

> **Note**: You can get the **ClientId** from the WebViewer control. By default, it is WebViewer1.

ViewerViewModel

To work with the API, first initialize the viewer's view model using the **GetWebViewer(clientId) function**. The clientId is the WebViewer control's name, by default, WebViewer1. If there is no ViewerViewModel with the requested clientId, an exception occurs.

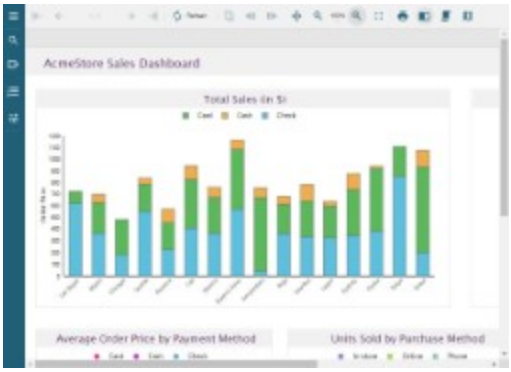
Use code like the following to initialize the ViewerViewModel:

```
var viewModel = GetWebViewer(clientId);
```

JavaScript Viewer

The JSViewer is a Javascript component that you can easily customize and use in web applications to preview all types of reports - page, rdl, and section reports. The JSViewer works on modern web application frameworks - ASP.NET MVC, ASP.NET Core MVC, and major JavaScript Frameworks such as Angular, React, and Vue.js.

A responsive visualization allows the JSViewer to automatically adjust to the screen size, so you can use it on all desktop, mobile and touch devices and modern browsers.

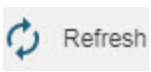


JSViewer UI

JSViewer Sidebar

Sidebar Element	Name	Description
	Display Sidebar	Displays the sidebar that includes the Search, Export, and Parameters panes.
	Search	Displays the Search pane.
	Export	Displays the Export pane where you can select an export format and options for the report you are previewing.
	Parameters	Displays the Parameters pane. If a report does not have parameters, the Parameters button is not displayed.
	Table of Contents	Displays the Table of Contents pane. If a report does not have the Label property or Document map label set, the Table of Contents or Documents map pane does not appear.

JSViewer Top Toolbar

Toolbar Element	Name	Description
	Go to First/Last page	Jumps to the first or last page of a report.
	Go to Previous/Next page	Navigates through a report page by page.
	Current page	Displays the current page number and page total. Enter the page number to view a specific page.
	Refresh	Refreshes the report.
	History: Back to Parent	Returns to the parent report in a drill-down page report or RDL report.
	History: Go Back	Navigates to a previous page in a parent report in a drill-down page report or RDL report.
	History: Go Forward	Navigates to a next page in a parent report in a drill-down page report or RDL report.
	Move Tool	A move tool that you can use to navigate the report.
	Zoom Out	Decreases the magnification of your report.
	Zoom mode	Displays the current zoom percentage which can also be edited. Allows you to select from the available zoom options - **50%**, **100%**, **150%**, **200%,** 300%, **Fit to Page**, and **Fit to**

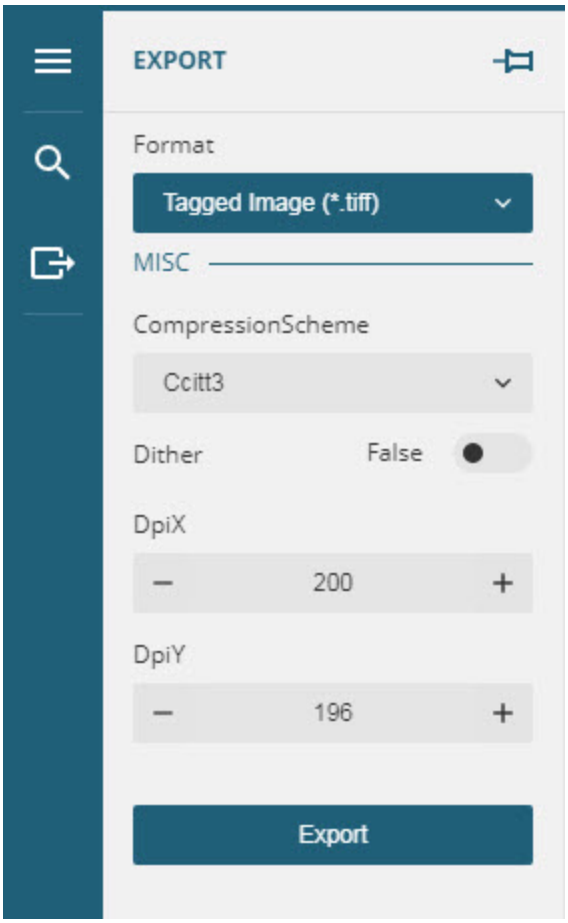
		Width.
	Zoom In	Increases the magnification of your report.
	Toggle Fullscreen	Switches to a Fullsreen mode.
	Print	Displays the Print screen to specify printing options. See Print in JSViewer topic for more information on printing.
	Single Page View	Shows one page at a time in the viewer.
	Continious View	Shows all preview pages one below the other.
	Galley mode	Provides a viewer mode which removes automatic page breaks from an RDL report and displays data in a single scrollable page. This mode maintains page breaks you create in the report.

JSViewer Search

The **Search** pane lets you enter a word or phrase for which to search within the report. Under **Use these additional criteria**, you may optionally select additional criteria. When you click **Search**, any results appear in the **Find results** list. Click an item in the list to jump to the item you selected and highlight it.

JSViewer Export

To display the Export pane, click **Export** in the sidebar. The **Export** pane lets you enter parameters for exporting a report that you are previewing. The available format options are Excel 2003, Excel, Word 2003, Word, PDF, CSV, JSON, XML, Tagged Image, and Web Archive.

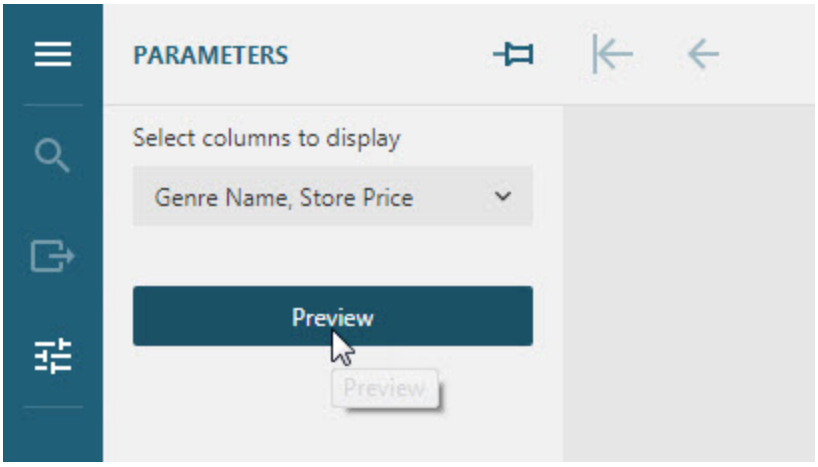


After you set all necessary export properties, click **Export**.

JSViewer Parameters Pane

The Parameters pane appears when you click the **Parameters** button in the sidebar. In the **Parameters** pane, enter a value to filter the data to be displayed and click **Preview**.

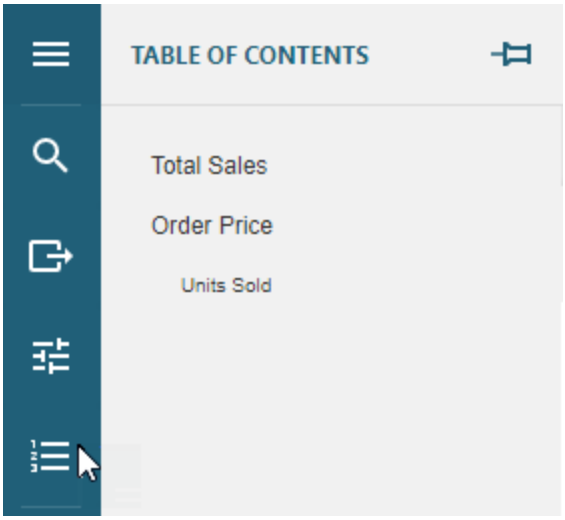
If a report does not have parameters, the Parameters pane is disabled.



JSViewer Table of Contents Pane

The Table of Contents pane appears when you click the **Table of Contents** button in the sidebar. Click any TOC item to navigate to the corresponding section of the report in the Viewer.

Note that the Table of Contents is only available for reports with Bookmarks. The Table of Contents displays each value for the text box, group, or subreport that you bookmark, and you can click them to navigate to the corresponding section of the report in the Viewer.



JSViewer API

Initialization Options

The following options can be set during initialization or at run time while working with the JSViewer.

action

Description: The callback that is invoked before the JSViewer opens the hyperlink, bookmark link, drill down report or toggles the report control visibility.

Type: function(actionType, actionParams)

Example:

```
action: (actionType, actionParams) => console.log('Action type: ' + actionType + '; Action parameters: ' + actionParams)
```

availableExports

Description: The array of export types available via Export functionality of JSViewer. By default, PDF, Excel, Word, JSON, XML, CSV, and Image exports are available.

Type: Array

Example:

```
availableExports: ['Xml', 'Pdf']
```

displayMode

Description: Set up single page or continuous page.

Type: String

Accepted Value: 'single', 'continuous'

Example:

```
displayMode: 'Continuous'
```

documentLoaded

Description: The callback that is invoked when a document is loaded entirely on the server.

Type: function()

Example:

```
documentLoaded: () => console.log('The document is loaded entirely on the server')
```

Returns: Void

element

Description: JQuery selector that specifies the element that hosts the JSViewer control.

Type: String

Example:

```
element: '#viewerContainer'
```

error

Description: The callback that is invoked when an error occurs in the process of displaying the report. The default error panel does not appear if the callback returns true. The error parameter is an object that has a message property which allows the users to customize the error message.

Type: function(error)

Example: To hide the default error panel:

```
error: (error) => {
    if(error.message) {
    // show error message.
    alert("Internal error! Please ask administrator.");
    // do not show default error message.
    }
}
```

locale

Description: Specifies predefined locale used for displaying the viewer. If locale is not specified explicitly here, the locale corresponding to the browser preferences is used. Localization can be done in three languages:

Type: String

Accepted Values: 'en-US' (for English), 'ja-JP' (for Japanese), and 'zh-CN' (for Chinese)

Example:

```
locale: 'ja-JP'
```

localeData

Description: The JSON containing the localization strings. All strings are not necessary - missing values are displayed using EN localization data. It is like `localeUri`, but instead of the path to localization data, localization data itself via JSON object is specified.

Type: JSON

Example:

```
localeData: JSON.parse( `{
    "export":  {
      "boolTextFalse": "いいえ",
      "boolTextTrue": "はい"
    },
    "viewer": {
      "toolbar": {
        "refresh": "更新"
      }
    }
  }`
),
```

localeUri

Description: The url of the file containing the localization strings (custom-locale.json file). Not all strings are necessary - missing values are displayed using EN localization data.

Type: String

Example:

```
localeUri: './custom-locale.json'
```

Content of custom-locale.json:

```
{
  "export": {
    "boolTextFalse": "いいえ",
    "boolTextTrue": "はい",
    ...
  },
  "exportcsv": {
    "friendlyName": "-CSV-",
    "settings": {
      "ColumnsDelimiter": {
        "label": "列の区切り",
        "category": "その他"
      },
      ...
    },
    ...
  },
  ...
}
```

renderMode

Description: Set up initial render mode: 'Paginated' or 'Galley'. Default value is 'Paginated'.

Type: String

Accepted Values: 'Galley', 'Paginated'

Example:

```
renderMode: 'Galley'
```

reportID

Description: The id of the report to be shown by the JSViewer.

Type: String

Example:

```
reportID: 'AnnualReport.rdlx'
```

reportLoaded

Description: The callback that is invoked when the JSViewer obtains the information about the requested report. The reportInfo object is passed in the callback including the TOC info, Parameters info and the link to the rendered report result.

Type: function(reportInfo)

Example:

```
reportLoaded: (reportInfo) => console.log('The report ' + reportInfo.name + ' was
successfully loaded!')
```

reportParameters

Description: The array of the {name, value} pairs that describe the parameters values used to run the report.

Type: Array

Example:

```
reportParameters: [{ name: 'ReportParameter1', values: ['1']}]
```

reportService

Description: Set up the settings to connect the Web API.

Type: Object that has the following optional properties:

- **url**: The url to connect the Web API.
  Type: String
  Example:
  - use prefix in url:
    ```
    url: '/api/reporting' // default value
    ```
  - use full URL:
    ```
    url: 'http:example.com/api/reporting'
    ```
- **securityToken**: The security key required to access the Web API.

Type: String
Example:
`securityToken: 'security_token'`

- **onRequest**: Callback before any request. Takes the **init** argument that allows you to add an option before fetch request, for example, to change the security token.
  Type: function
  Example:
  `onRequest: (init) => init.headers.Authorization = 'security_token'`

  The **init** argument is an object that takes the following options:

  - **credentials**: Set request's credentials.
  - **headers**: Set request's headers.
  - **signal**: Set request's signal.

  Please see https://fetch.spec.whatwg.org/#requestinit for more information.

Example:

```
reportService: {
    url: 'http://example.com/api/reporting', //web service url
    securityToken: 'security_token', //provide securityToken
    onRequest: (init) => {
         init.credentials = "include",
         init.headers = { "Cache-Control": "no-cache, no-store, must-revalidate",
"Expires": "0", "Pragma": "no-cache", "Accept": "application/json" },
         init.signal = new AbortController().signal
         }
    }
```

Public API Methods and Properties

Methods

backtoParent

Description: Makes the viewer to display the parent report of the drill-down report.

Syntax: `backToParent()Void`

Example:

```
viewer.backToParent()
```

Return Value: Void

create

Description: Creates a new instance of the Viewer and renders it in the specified DOM element.

Syntax: `create()Void`

Parameters:

- params

Example:

```
const viewer = GrapeCity.ActiveReports.JSViewer.create({
    element: '#root',
    reportID: 'AnnualReport.rdlx',
    availableExports: ['Xml', 'Pdf'],
    // other properties
});
```

Return Value: Void

destroy

Description: Removes the viewer content from the element.

Syntax: `destroy()Void`

Example:

```
viewer.destroy()
```

Return Value: Void

export

Description: Exports the currently displayed report.

Syntax: `export(exportType,callback,saveAsDialog,settings)Void`

Parameters:

- **exportType**: Specifies export format.
- **settings**: The export settings are available for RenderingExtensions.
- **Note:** In section reports, the export settings are not enabled when exporting files using the rendering extensions. In Page report and RDL report, the export settings are not enabled when exporting files to PDF using the export filter.

Example:

```
viewer.export('Pdf', downloadReport, true, { Title: '1997 Annual Report' })
```

Return Value: Void

getTOC

Description: Obtains the report TOC.

Syntax: viewer.getToc()

Example:

```
console.log(viewer.getToc())
```

Return Value: any

goToPage

Description: Makes the viewer to display the specific page.

Syntax: `goToPage(number, offset, callback)Void`

Parameters:

- **number**: The number of pages to go to.

Example:

```
viewer.goToPage(1)
```

Return Value: Void

openReport

Description: Opens a report.

Syntax: `openReport(reportID: string, reportParameters?: Array<Parameter>): void`

Parameters:

- **reportID**: The id of the report.

Example:

```
viewer.openReport('Invoice.rdlx')
```

Return Value: Void

print

Description: Prints the currently displayed report if any.

Syntax: `print()Void`

Example:

```
viewer.print()
```

Return Value: Void

refresh

Description: Refreshes the report preview.

Syntax: `option(name,[value])Object`

Example:

```
viewer.refresh()
```

Return Value: Void

version

Description: A string that represents the current version of the JSViewer.

Syntax: `GrapeCity.ActiveReports.JSViewer.version`

Example:

```
console.log(GrapeCity.ActiveReports.JSViewer.version)
```

Return Value: string

toolbar.addItem

Description: Adds a custom toolbar item (button) to the toolbar's end.

Type: item: ToolbarItem

Example:

```
ToolbarItem = {
    key: string;
    iconCssClass?: string;
    icon?: Icon;
    text?: string;
    title?: string;
    checked?: boolean;
    enabled?: boolean;
    action?: (toolbarItem) => function;
    onUpdate?: (args: ChangedEventArgs, toolbarItem) => function;
}

Icon = FontIcon | SVGIcon;


SVGIcon = {
    type: 'svg';
    content: any; //(string | JSX.element)
    size?: Size;
    rotate?: '90' | '180' | '270';
};


FontIcon = {
    type: 'font';
    iconCssClass: string;
    fontSize?: string;
}

var pdfExportButton = {
    key: '$pdfExportButtonKey',
    iconCssClass: 'mdi mdi-file-pdf',
    enabled: true,
    action: function(item) {
        console.log('Export to PDF function works here');
    },
    onUpdate: function(arg, item) {
        console.log('Something in viewer was updated, check/update button state here');
    }
};


//OR button with svg as icon
var icon = `

`;
```

```
var svgPdfExportButton = {
    key: '$pdfExportButtonKey',
    icon: {type: 'svg', content: icon},
    enabled: true,
    action: function(item) {
        console.log('Export to PDF function works here');
    },
    onUpdate: function(arg, item) {
        console.log('Something in viewer was updated, check/update button state here');
    }
}


viewer.toolbar.addItem(pdfExportButton);
//or
viewer.toolbar.addItem(svgPdfExportButton);
```

toolbar.updateItem

Description: Updates the layout/function/state for a specified single item.

Type: String: key, ToolbarItem item

Example:

```
var xlsxExportButton = {
    key: '$xlsxExportButtonKey',
    iconCssClass: 'mdi mdi-file-pdf',
    enabled: true,
    action: function(item) { console.log('Export to XLSX function works here'); },
    onUpdate: function(arg, item) { console.log('Something in viewer was updated,
check/update button state here'); }
};
viewer.toolbar.updateItem('$pdfExportButtonKey', xlsxExportButton); //from now button
will behave as described in new properties
```

toolbar.removeItem

Description: Removes a custom toolbar item from the toolbar. To remove the default one, you should use the **updateLayout** function.

Type: String: key

Example:

```
viewer.toolbar.removeItem('$pdfExportButtonKey');
```

toggleSidebar

Description: Toggles the sidebar panel (hides or shows it). If no parameter is passed, works as toggle.

Type: boolean

Example:

```
viewer.toggleSidebar();//hide sidebar
viewer.toggleSidebar(true);//show sidebar
```

Properties

currentPage

Description: Gets the currently displayed page number.

Syntax: `viewer.currentPage`

Example:

```
console.log(viewer.currentPage())
```

Return Value: An integer representing currently displayed page number.

pageCount

Description: Gets the page count of the currently displayed report.

Syntax: `viewer.pageCount`

Example:

```
console.log(viewer.pageCount())
```

Return Value: An integer representing page count.

Configure JSViewer

The ActiveReports 14 provides **JSViewer MVC** and **JSViewer MVC Core** templates in Visual Studio. Use the template to obtain a pre-configured application.

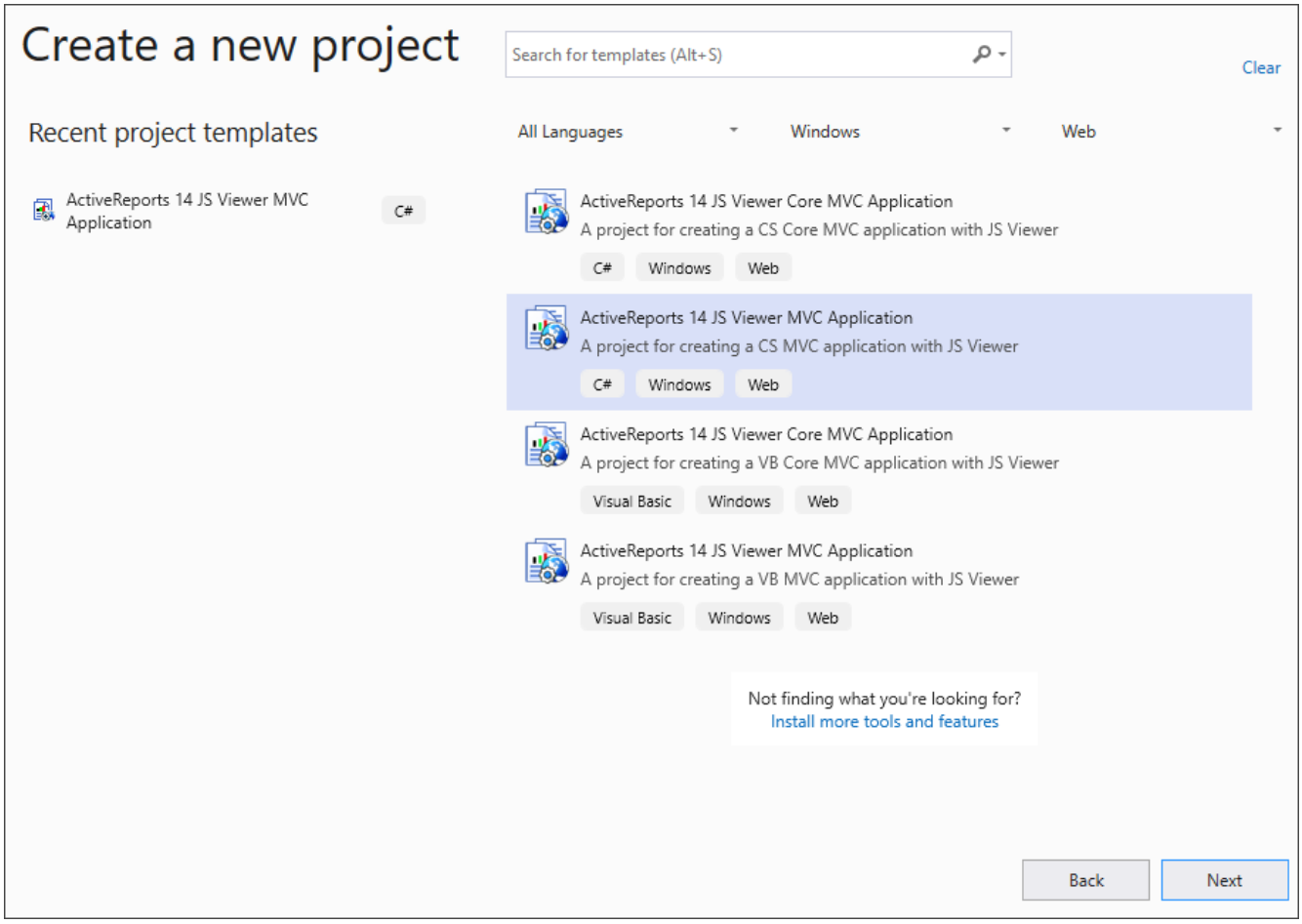
> **Note**: JSViewer Core templates are available only in Visual Studio 2019. Visual Studio 2013, Visual Studio 2015, and Visual Studio 2017 do not support the JSViewer Core templates.

Prerequisites

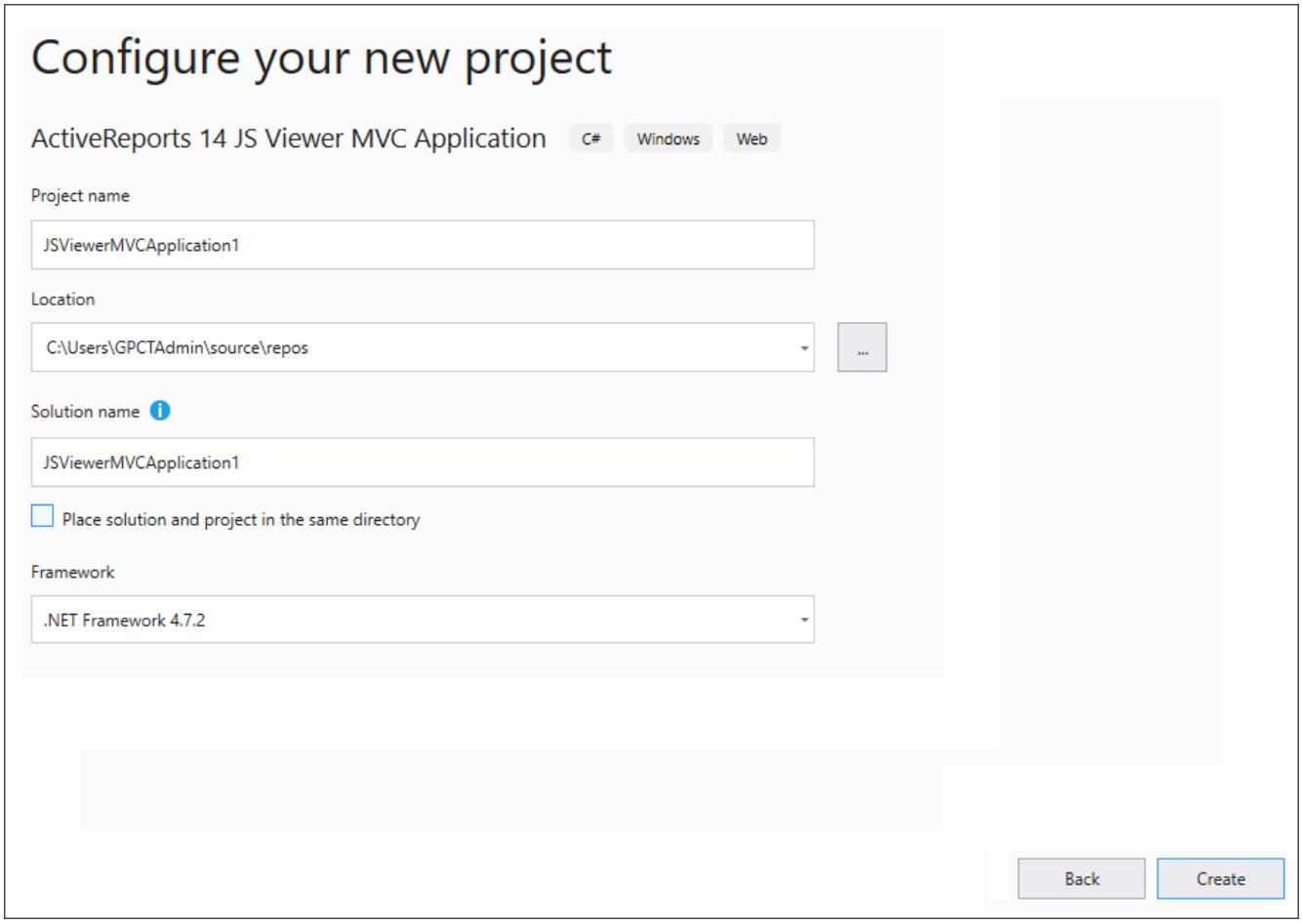
- ActiveReports 14 must be installed.
- Node.js must be installed.

The following steps describe how to use a JSViewer MVC template in Visual Studio 2019:

1. Open **Microsoft Visual Studio 2019** and create a new **ActiveReports 14 JSViewer MVC Application** project.



2. On 'Configure your new project' page, fill-in the details such as Project name - JSViewerMVCApplication, Framework - .NET Framework 4.7.2, and select **Create**.



The ActiveReports 14 JSViewer template automatically does the following:

- adds NuGet packages necessary for viewer
- creates RDL report in Reports folder.
- configures index.html
- adds script and CSS files

You can also download and install the JSViewer-related files and folders from NPM using the following command in the command line:

```
npm install @grapecity/ar-viewer
```

The viewer files/folders are downloaded in your current directory: .*node_modules**@grapecity**ar-viewer**dist*

3. Copy the report you want to preview in the 'Reports' folder.
4. Open index.html and provide the name of the report in **viewer.openReport()** method as:

```
viewer.openReport("Reportname.rdlx");
```

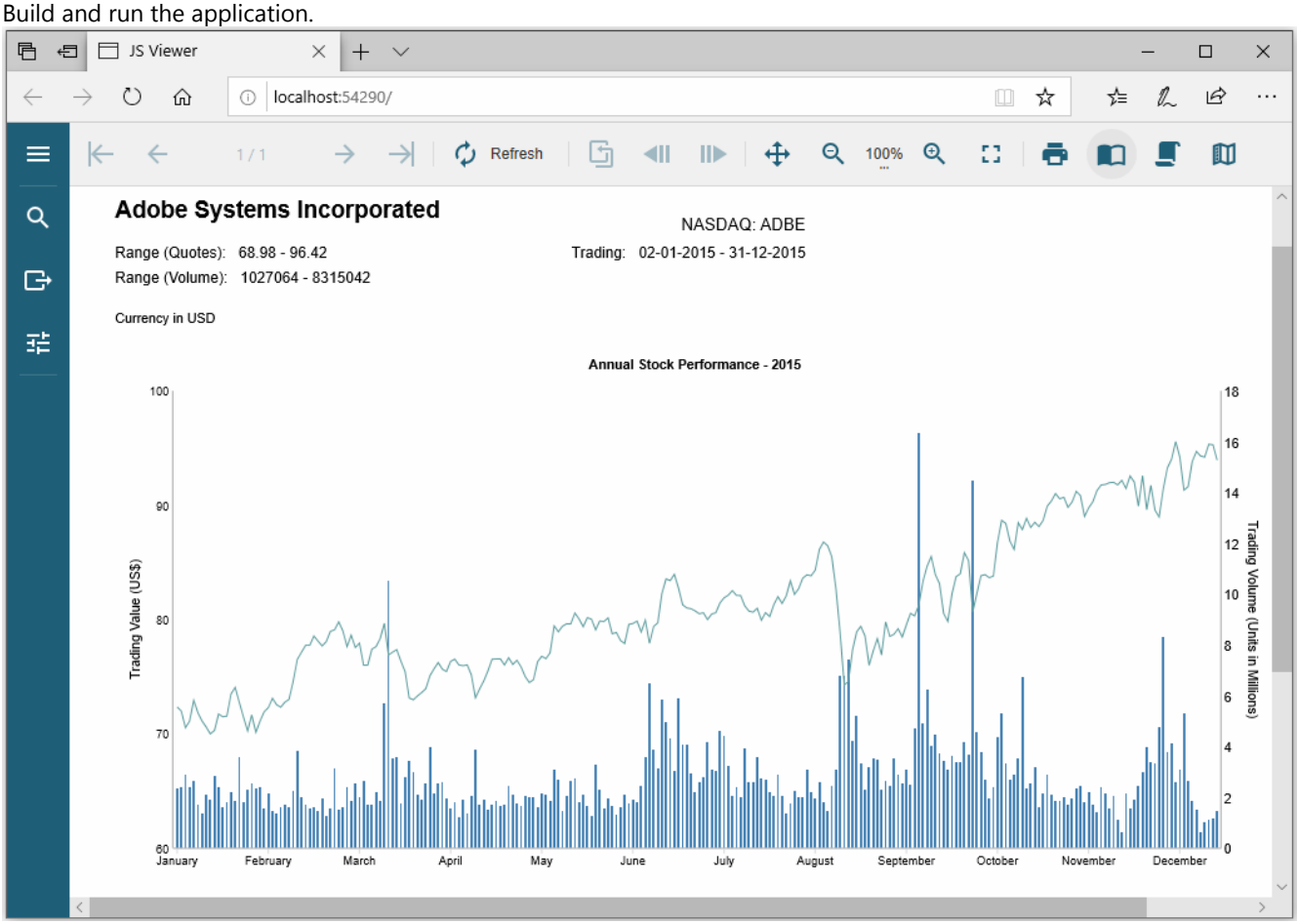
5. Modify the **Startup.cs** file to include the path of the report folder.

Startup.cs

```
public class Startup
  {
      public static string EmbeddedReportsPrefix = "JSViewerMVCApplication2.Reports";
      public void Configuration(IAppBuilder app)
```

```
        {
            app.UseErrorPage();
            app.UseReporting(settings =>
            {
                settings.UseEmbeddedTemplates(EmbeddedReportsPrefix,
Assembly.GetAssembly(GetType()));
                settings.UseCompression = true;
                settings.UseFileStore(new System.IO.DirectoryInfo("path to the Reports
folder"));
            });
            RouteTable.Routes.RouteExistingFiles = true;
        }
    }
```

6. Build and run the application.



Previewing Reports in JSViewer

The **JSViewer** when used in MVC and MVC Core applications, allows you to preview all the following file formats:

- .rdl
- .rdlx
- .rpx

- .rdlx-master

If you want to view Code-Based Section reports (.cs or .vb), or RDF files (.rdf) you need to follow a little different approach as elaborated in following sections.

> **Note**:
>
> - Previewing Section reports and RDF files is **not** supported in **Core** applications.
> - The WebViewer and JSViewer are supported only in the **Integrated pipeline mode**. You will get PlatformNotSupportedException on using these Viewers in Classic pipeline mode.

To preview Code-Based Section reports in JSViewer

To view Code-Based Section reports (file format .cs or .vb), you need to do few modifications in your code provided in Configure JSViewer topic as follows:

1. Modify the **startup.cs** file as:

Startup.cs

```
public class Startup
    {
        public static string EmbeddedReportsPrefix = "'JSViewerMVCApplication1.Reports";
        public void Configuration(IAppBuilder app)
        {
            app.UseErrorPage();
            app.UseReporting(settings =>
            {
                settings.UseEmbeddedTemplates(EmbeddedReportsPrefix,
Assembly.GetAssembly(GetType()));
                settings.UseCompression = true;
                settings.UseCustomStore(GetReport);
            });
            RouteTable.Routes.RouteExistingFiles = true;
        }
        public object GetReport(string reportName = "SectionReport")
        {
            SectionReport1 rpt = new SectionReport1();
            return rpt;
        }
    }
```

Startup.vb

```
Public Class Startup
    Public Sub Configuration(ByVal app As IAppBuilder)

        app.UseReporting(Function(settings)
                             settings.UseCompression = True
                             settings.UseCustomStore(AddressOf GetReport)
                             Return settings
```

```
                        End Function)

    End Sub

    Private Function GetReport() As Object
        Dim rpt As New SectionReport1()
        Return rpt
    End Function
```

2. Update **index.html**:

index.html

```
<html lang="en">
<head>
  <meta charset="utf-8">
  <link rel='shortcut icon' type='image/x-icon' href='favicon.ico' />
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <meta name="theme-color" content="#000000">
  <title>JSViewer</title>
  <link href="jsViewer.min.css" rel="stylesheet">
  <link href="index.css" rel="stylesheet">
</head>
<body onload="loadViewer()">
  <div style="width: 100%; overflow-x: hidden">
    <div style="float:right;width:100%" id="viewerContainer">
    </div>
  </div>
  <script type="text/javascript" src="jsViewer.min.js"></script>
  <script type="text/javascript">
    let viewer;
    function loadViewer() {
      viewer = GrapeCity.ActiveReports.JSViewer.create({
          element: '#viewerContainer',
          reportID: 'JSViewerMVCApplication1.Reports.SectionReport1'
      });
    }
  </script>
</body>
</html>
```

To preview RDF files in JSViewer

To view RDF files (file format .rdf), you need to do few modifications in your code provided in Configure JSViewer topic as follows:

1. Update **HomeController.cs** to include 'rdf' as valid extension:

HomeController.cs

```
string[] validExtensions = { ".rdl", ".rdlx", ".rdlx-master", ".rpx", ".rdf" };
```

HomeController.vb

```
Dim validExtensions = {".rdl", ".rdlx", ".rdlx-master", ".rpx", ".rdf"}
```

2. Update **index.html** to include report name in **viewer.openReport()** method:

index.html

```
<html lang="en">
<head>
  <meta charset="utf-8">
  <link rel='shortcut icon' type='image/x-icon' href='favicon.ico' />
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <meta name="theme-color" content="#000000">
  <title>JSViewer</title>
  <link href="jsViewer.min.css" rel="stylesheet">
  <link href="index.css" rel="stylesheet">
</head>
<body onload="loadViewer()">
  <div style="width: 100%; overflow-x: hidden">
    <div style="float:right;width:100%" id="viewerContainer">
    </div>
  </div>
  <script type="text/javascript" src="jsViewer.min.js"></script>
  <script type="text/javascript">
    let viewer;
    function loadViewer() {
      viewer = GrapeCity.ActiveReports.JSViewer.create({
          element: '#viewerContainer'
      });
     viewer.openReport("RdfReport.rdf");
    }
  </script>
</body>
</html>
```

WPF Viewer

ActiveReports provides the WPF Viewer that you can use to load and view your reports. This viewer contains a toolbar and a sidebar with **Thumbnails**, **Search results**, **Document map** and **Parameters** panes.

The WPF Viewer has a limitation in the continuous mode. In this mode, a report is always rendered on top of the WPF Viewer because the WinForms content always appears on top of the WPF content.

> ⚠ **Warning:** The TargetInvocationException occurs on running a WPF browser application in Partial Trust. Refer to **Running WPF Viewer in Partial Trust** section given below to run your WPF Browser Application in Partial Trust. To

ensure that you are using Full Trust:

1. From the Visual Studio **Project** menu, select **YourProject Properties**.
2. On the **Security** tab, under **Enable ClickOnce security settings**, select the **This is a full trust application** option.

WPF Viewer Toolbar

The following table lists the actions you can perform through the WPF Viewer toolbar.

Toolbar Element	Name	Description
	Toggle Sidebar	Displays the sidebar that includes the Thumbnails, Parameters, Document map and Search results panes.
	Print	Displays the Print dialog where you can specify the printing options.
	Galley mode	Provides a viewer mode which removes automatic page breaks from an RDL report and displays the data in a single scrollable page. This mode maintains page breaks you create in the report.1
	Find	Displays the Find dialog to find any text in the report.
	Zoom out	Decreases the magnification of your report.
	Zoom in	Increases the magnification of your report.
	Current zoom	Displays the current zoom percentage which can also be edited.
	Fit page width	Fits the width of the page according to viewer dimensions.
	Fit whole page	Fits the whole page within the current viewer dimensions.
	First page	Takes you to the first page of the report. This button is enabled when a page other than the first page is open.
	Previous page	Takes you to the page prior to the current page. This button is enabled when a page other than the first page is open.
	Current page	Opens a specific page in the report. To view a specific page, type the page number and press the Enter key.
	Next page	Takes you to the page following the current page. This button is disabled on reaching the last page of

		the report.
	Last page	Takes you to the last page of the report. This button is disabled on reaching the last page of the report.
	Backward	Takes you to the last viewed page. This button is enabled when you move to any page from the initial report page. Clicking this button for the first time also enables the **Forward** button.
	Forward	Takes you to last viewed page before you clicked the Backward button. This button is enabled once you click the **Backward** button.
	Back to parent report	Returns you to the parent report in a drillthrough report.
	Refresh	Refreshes the report.
	Cancel	Cancels rendering of the report.

[1]To view drillthrough reports in WPF Viewer, you need to turn off the Galley mode.

WPF Viewer Sidebar

The WPF Viewer sidebar appears on the left of the Viewer control when you click the **Toggle sidebar** button in the toolbar. By default, this sidebar shows the Thumbnails and Search Results panes. The additional Document map and Parameters also appear in this sidebar. You can toggle between any of the viewer panes by clicking the buttons for each pane at the bottom of the sidebar.

Thumbnails pane

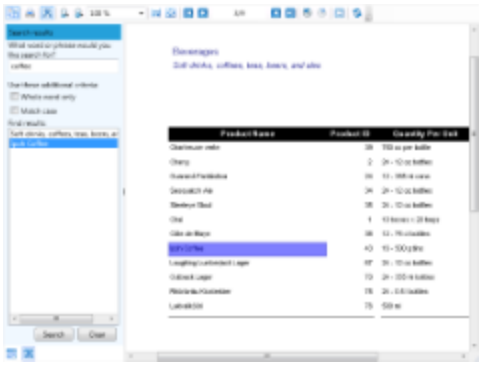
The **Thumbnails** pane appears by default in the sidebar when you click the **Toggle sidebar** button in the toolbar.

This pane is composed of a thumbnail view of all the pages in a report. Click any thumbnail to navigate directly to the selected report page. You can also modify the size of the thumbnail when you click (+) or (-) button to zoom in and zoom out.

Search results pane

The **Search** pane is the other default pane besides Thumbnails that appears in the sidebar when you click the **Toggle**

sidebar button. This pane lets you enter a word or phrase from which to search within the report.



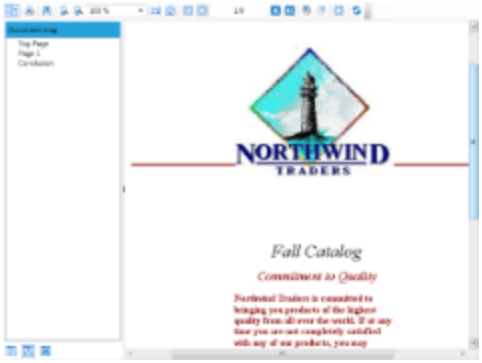
To search in a report:

- Enter the word or phrase in the search field.
- Under **Use these additional criteria**, you may optionally choose to search for the whole word or match the case of the search string while searching in the report.
- Click the **Search** button to see the results appear in the **Find results** list.
- Click an item in the list to jump to that item in the report and highlight it.

To start a new search or clear the current search results, click the **Clear** button under the **Find results** list.

Document map pane

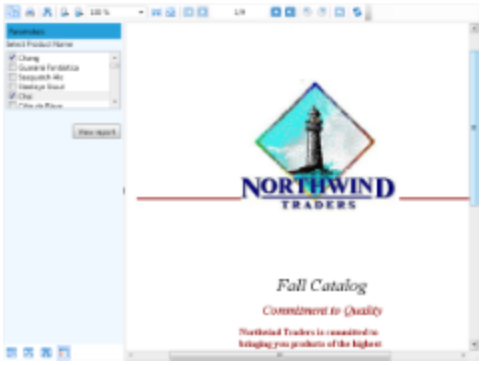
The Documents map pane sis enabled for reports where the Label property or the Document map label is set. This pane displays each value for the text box, group, or sub report that you label, and you can click them to navigate to the corresponding area of the report in the Viewer.



If a report does not have the Label property or Document map label set, the Documents map panes does not appear in the sidebar.

Parameters pane

The WPF Viewer allows you to view reports with parameters. In the toolbar, click the **Toggle sidebar** button to open the WPF Viewer sidebar and if your report contains parameters, the **Parameters** pane shows up automatically.



1. In the **Parameters** pane, you are prompted to enter a value by which to filter the data to display.
2. Enter a value or set of values and click **View report**, to filter the report data and display the report.

If a report does not have parameters, the Parameters pane does not appear in the sidebar.

Additional Features

Following is an introduction to the additional capabilities of the Viewer to guide you on using it effectively.

Keyboard Shortcuts

The following shortcuts are available on the WPF Viewer.

Keyboard Shortcut	Action
Ctrl + F	Shows the search pane.
Ctrl + P	Shows the print dialog.
Esc	Closes the print dialog.
Page Down	Moves to the next page.
Page Up	Moves to the previous page.
Ctrl + T	Shows or hides the table of contents.
Ctrl + Home	Moves to the first page.
Ctrl + End	Moves to the last page.
Ctrl + Right	Navigates forward.
Ctrl + Left	Navigates backward.
Ctrl + -	Zooms out.
Ctrl + +	Zooms in.
Left, Right, Up, Down	Moves the visible area of the page in the corresponding direction.
Ctrl + 0 (zero)	Sets the zoom level to 100%.
Ctrl + rotate mouse wheel	Changes the zoom level up or down.

Ctrl + G	Moves the focus to current page toolbar option.
F5	Refreshes the report.

Advance Printing Options

You can set the **PrintingSettings ('PrintingSettings Property' in the on-line documentation)** property of the Viewer to directly print without displaying a dialog or to switch from a ActiveReports print dialog to a .NET Framework standard print dialog (System.Windows.Forms.PrintDialog) on clicking the **Print** button on the Viewer toolbar. You can set the **PrintingSettings** property of the Viewer control from the Properties window.

PrintingSettings property provides the following options:

PrintingSettings Options	Description
ShowPrintDialog	Displays a dialog in which the user can set the printer options before printing.
ShowPrintProgressDialog	Displays a print progress dialog, in which the user can cancel the printing job.
UsePrintingThread	Specifies whether printing should be performed for individual threads or not.
UseStandardDialog	Specifies whether to use the .NET Framework standard print dialog (System.Windows.Forms.PrintDialog) while printing the document (section or page).

Running WPF Viewer in Partial Trust

You can run a WPF Viewer Browser Application in Partial Trust by customizing the permission set in the app.mainfest file. While customizing the permission set, some of the permissions are to be set explicitly like MediaPermission, UIPermission, etc. Follow these steps to learn how to run a WPF Viewer application in partial trust.

> **Note**: These steps assume that you have already created a WPF Browser Application that contains the WPF Viewer control.

1. In the **Solution Explorer**, right-click your *WPF Browser Application* and select **Properties**.
2. On the properties page that opens, go to the **Security** tab.
3. On the Security tab, set **Zone your application will be installed from** to Custom and click the **Edit Permissions XML** button. This opens the app.mainfest file.
4. In the app.manifest file, replace the existing xml code inside the **<applicationRequestMinimum> </applicationRequestMinimum>** tags with the following code.

   Code

   XML

```
<defaultAssemblyRequest permissionSetReference="Custom" />
<PermissionSet class="System.Security.PermissionSet" version="1" ID="Custom"
SameSite="site">
  <IPermission class="System.Data.SqlClient.SqlClientPermission, System.Data,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" version="1"
Unrestricted="true" />
```

```
  <IPermission class="System.Data.OleDb.OleDbPermission, System.Data,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" version="1"
Unrestricted="true" />
  <IPermission class="System.Security.Permissions.SecurityPermission, mscorlib,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" version="1"
Unrestricted="true" />
  <IPermission class="System.Security.Permissions.MediaPermission, WindowsBase,
Version=3.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" version="1"
Audio="SafeAudio" Video="SafeVideo" Image="SafeImage" />
  <IPermission class="System.Security.Permissions.UIPermission, mscorlib,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" version="1"
Unrestricted="true" />
  <IPermission class="System.Security.Permissions.EnvironmentPermission, mscorlib,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" version="1"
Unrestricted="true" />
  <IPermission class="System.Security.Permissions.FileIOPermission, mscorlib,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" version="1"
Unrestricted="true" />
  <IPermission class="System.Security.Permissions.ReflectionPermission, mscorlib,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" version="1"
Unrestricted="true" />>
  <IPermission class="System.Security.Permissions.StrongNameIdentityPermission, ,
mscorlib, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
version="1" Unrestricted="true" />
</PermissionSet>
```

5. Press F5 to run your *WPF Browser Application*.

Note: The Partial Trust Limitations are applicable to the WPF viewer running in Partial Trust. For more information, see Medium Trust Support.

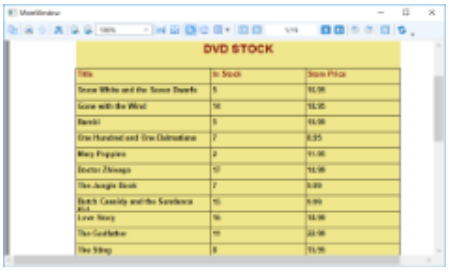
View Reports in WPF Viewer

The ActiveReports WPF Viewer is a custom control that allows to easily view section, RDL and page report layouts.

This walkthrough is split up into the following activities.

- Creating a WPF Application project in Visual Studio
- Adding the ActiveReports WPF Viewer control to the xaml page
- Loading a report to the ActiveReports WPF Viewer
- Previewing a report
- Customizing the ActiveReports WPF Viewer

When you have completed these steps, you will have the ActiveReports WPF Viewer displaying a report that looks similar to the following.



To create a WPF application in Visual Studio

1. On the Visual Studio **File** menu, click **New** > **Project**.
2. In the **New Project** dialog that appears, select **WPF Application** in the list of templates.
3. Specify the name and location of the project and then click the **OK** button.
4. In the Visual Studio Solution Explorer, right-click *YourProject* and select **Add**, then **New Item**.
5. In the **Add New Item** dialog that appears, select the **ActiveReports 14 Page Report** and create the rptSingleLayout report as described in the Single Layout Reports walkthrough.

To add the WPF Viewer control

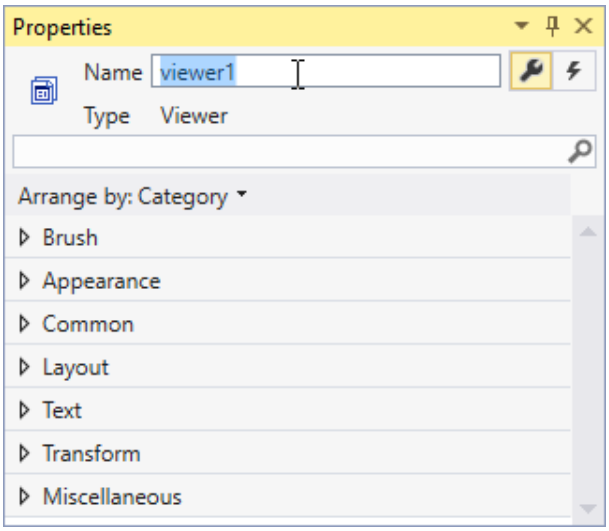
1. In Solution Explorer, open MainWindow.xaml.
2. From the Toolbox ActiveReports 14 tab, drag the **Viewer** control and drop it on the design view of MainWindow.xaml.

   > **Note:** Dragging the **Viewer** control to the design view of MainWindow.xaml automatically adds the corresponding reference to the licenses.licx file.

3. In the **Properties** window, set the properties of the Viewer control as follows.

Property Name	Property Value
HorizontalAlignment	Stretch
VerticalAlignment	Stretch
Margin	0

4. In the **Properties** window, rename the Viewer control to **viewer1**.



To load a report to the WPF Viewer

1. In the Solution Explorer, select the rptSingleLayout report you have created**.**

2. In the Properties window, set **Copy to Output Directory** to **Copy Always**.

> **Note:** If Page/RDL report is selected from dropdown of Source property (containing relative path), 'Copy to Output Directory' for selected report should be set as 'Copy always/Copy if newer' otherwise error "Could not find file ... " appears on loading WPF Viewer.

> **Caution:** In WPF Viewer control, previewing code-based Section reports using **Source ('Source Property' in the on-line documentation)** and **SourceFormat ('SourceFormat Property' in the on-line documentation)** properties is not supported.

3. On MainWindow.xaml, with the viewer selected, go to the Properties window and double click the Loaded event.
4. In the MainWindow code view that appears, add code like the following to the viewer1_loaded event to bind the report to the viewer. This code shows an .rdlx report being loaded, but you can use an .rpx report as well.

Visual Basic.NET code. Paste INSIDE the viewer1_Loaded event in MainWindow.xaml.vb.

```
Viewer1.LoadDocument("rptSingleLayout.rdlx")
```

C# code. Paste INSIDE the viewer1_Loaded event in MainWindow.xaml.cs.

```
viewer1.LoadDocument("rptSingleLayout.rdlx");
```

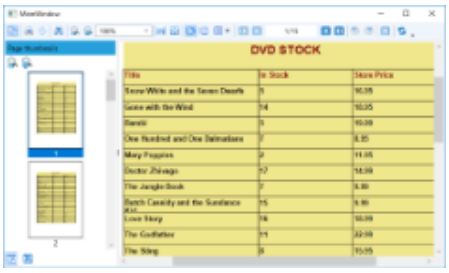
> **Note:**
>
> - Refer to the **LoadDocument ('LoadDocument Method' in the on-line documentation)** method to see other ways to load a report in WPF Viewer.
> - We can set report for WPFViewer directly on XAML file and load the report in WPF Viewer using **Source ('Source Property' in the on-line documentation)** and **SourceFormat ('SourceFormat Property' in the on-line documentation)** properties.
> - To avoid evaluation banners appearing at run time, license your ActiveReports WPF Application project. You can find information on licensing in License Your ActiveReports.

To view the report

Press F5 to run the project. The WPF Viewer displaying a report appears.

To customize the WPF Viewer

The ActiveReports WPF Viewer is a customizable control. You can easily change the look of the WPF Viewer and its elements, such as the **error panel**, **search panel**, **sidebar** and **toolbar** by modifying properties in the default WPF Viewer template (DefaultWPFiewerTemplates.xaml).



To add the customization template to the WPF project

1. Open your WPF project.

2. In Solution Explorer, select the *YourProjectName* node.
3. On the Visual Studio **Project** menu, click **Add Existing Item**.
4. In the dialog that appears, locate and select DefaultWPFViewerTemplates.xaml and click **OK**. You can find DefaultWPFViewerTemplates.xaml at [*systemdrive*]\Program Files\GrapeCity\ActiveReports 14\Deployment\WPF\Templates folder (on a **64-bit Windows operating system**, this file is located in [*systemdrive*]\Program Files (x86)\GrapeCity\ActiveReports 14\Deployment\WPF\Templates).
5. On MainWindow.xaml before the opening <Grid> tag, add the following code.

Paste to the XAML view of MainWindow.xaml before the opening <Grid> tag

```
<Window.Resources>
<ResourceDictionary Source="DefaultWPFViewerTemplates.xaml" />
</Window.Resources>
```

To customize the WPF Viewer sidebar

1. In Solution Explorer, double-click DefaultWPFViewerTemplates.xaml.
2. In the file that opens, search for **"thumbnails tab"**.
3. In the **GroupBox Header** property of <!-- thumbnails tab -->, remove "{Binding Source.ThumbnailsPane.Text}" and type **"THUMBNAILS"**.
4. Search for **"TabControl x:Name="Sidebar"**.
5. Add **Background="Yellow"** after TabControl x:Name="Sidebar".
6. Press **F5** to see the customized viewer sidebar.

To add a customized button to the WPF Viewer toolbar

1. In Solution Explorer, select the *YourProjectName* node.
2. On the Visual Studio Project menu, select **Add New Item**.
3. In the **Add New Item** dialog that appears, select **Class**, rename it to **MyCommand** and click **Add**.
4. In the MyCommand.cs/vb that opens, add the following code to implement a command.

To write the code in Visual Basic.NET

Visual Basic.NET code. Add to MyCommand.vb

```
Implements ICommand
Public Function CanExecute(ByVal parameter As Object) As Boolean Implements
System.Windows.Input.ICommand.CanExecute
    Return True
End Function

Public Event CanExecuteChanged(ByVal sender As Object, ByVal e As System.EventArgs)
Implements System.Windows.Input.ICommand.CanExecuteChanged


Public Sub Execute(ByVal parameter As Object) Implements
System.Windows.Input.ICommand.Execute
    MessageBox.Show("GrapeCity is the world's largest component vendor.", "About Us",
MessageBoxButton.OK)
End Sub
```

To write the code in C#

C# code. Add after the statement using System.Text;

```
using System.Windows.Input;
```

```
using System.Windows;
```

C# code. Add to MyCommand.cs

```
public class MyCommand : ICommand
    {
        public bool CanExecute(object parameter)
        {
            return true;
        }

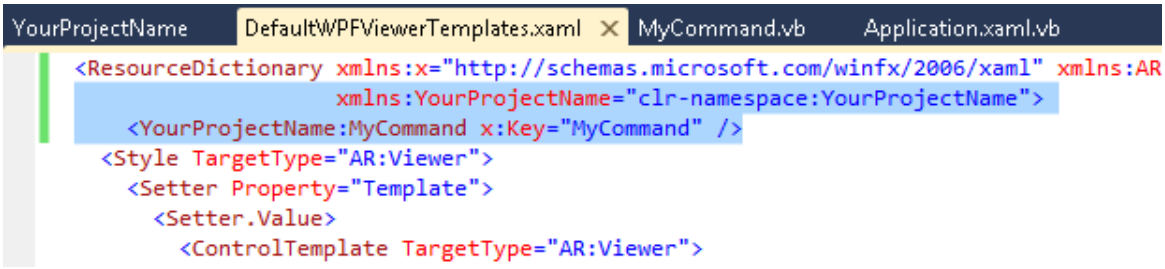
        public void Execute(object parameter)
        {
            MessageBox.Show("GrapeCity is the world's largest component vendor.", "About
Us", MessageBoxButton.OK);
        }

        public event EventHandler CanExecuteChanged;
    }
```

5. In Solution Explorer, double-click DefaultWpfViewerTemplates.xaml.
6. In the file that opens, add the following code.

XML code. Add to DefaultWpfViewerTemplates.xaml

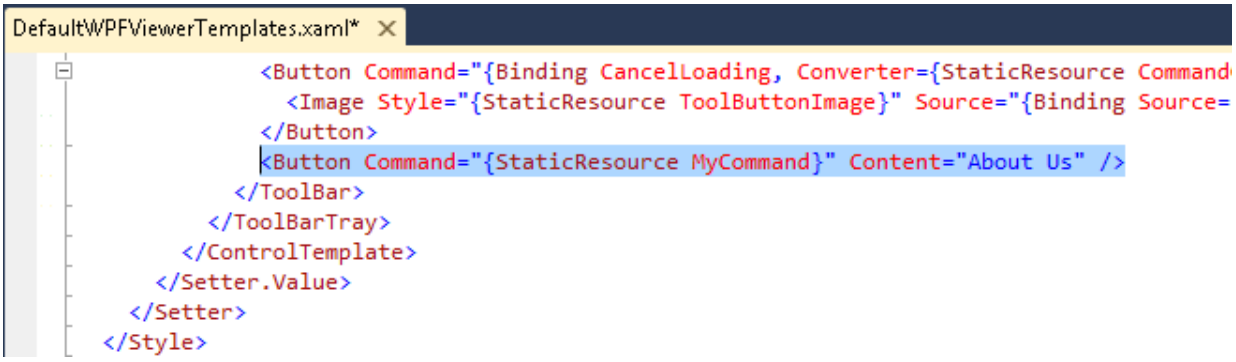
```
<ResourceDictionary>
 ...
xmlns:YourProjectName="clr-namespace:YourProjectName">
<YourProjectName:MyCommand x:Key="MyCommand" />
...
</ResourceDictionary>
```



7. In the same file, add the following code to add a button.

XML code. Add to DefaultWpfViewerTemplates.xaml before the closing Toolbar tag

```
<Button Command="{StaticResource MyCommand}" Content="About Us" />
```



8. Press F5 to see the new customized button **About Us** in the Viewer toolbar.

To remove the Refresh button from the WPF Viewer toolbar

1. In Solution Explorer, double-click DefaultWpfViewerTemplates.xaml.
2. In the file that opens, search for **"<!--Refresh btn-->"**.
3. Replace the existing content in **Visiblity="..."** with the following.

XML code. Add to DefaultWpfViewerTemplates.xaml
`<Button Command=... Visibility="Collapsed">`

Medium Trust Support

All features of ActiveReports are available without restrictions in a Full trust environment. You can also use ActiveReports under Medium trust, but with limitations on some of the features.

> ⚠ **Caution**: Medium trust does not adequately protect your application and should not be used. For more information see MSDN link.

> **Note**:
>
> - Assemblies placed in the Global Assembly Cache, or GAC (C:\WINDOWS\ASSEMBLY), have Full trust permissions, so the results on your deployment machine may differ from those on your development machine.
> - For information on licensing a class library project, see the article on Licensing a Project.

Feature Limitations

1. Exporting
   - RTF, Text, TIFF and Excel filters are not supported in Medium trust.
   - Digital signatures cannot be used in case of PDF rendering extension and PDF export.
   - Chart and Tablix control of Page Reports and RDL Reports are not displayed properly in case of PDF rendering extension and PDF export.
2. The End User Designer and Windows Form Viewer controls require Full trust.
3. The Picture control does not support **metafiles**, which require Full trust.
4. The **ImageType** property of the Chart control must be set to **PNG**.
5. OleObject, Formatted Text and Custom controls require Full trust.
6. Scripting requires Full trust, so if you need to use code in reports under Medium trust, use code-based reports rather than RPX format.
7. WebViewer
   - Bullet and Sparkline controls of PageReports and RDLReports are not displayed in all the Viewer types.

- Report containing the Image control of PageReports and RDLReports is not displayed properly in case of HTML type.

Recommended Development Environment for Medium Trust Tests

To set up a Medium trust environment

Open the Web.config file and paste the following code between the <system.web> and </system.web> tags.

XML code. Paste BETWEEN the system.web tags.

```
<trust level="Medium"></trust>
```

To set up the PrintingPermission level

Most hosting providers disable the printing permissions in a partially trusted environment.

1. Open the web_mediumtrust. config file (located in the \Windows\Microsoft.NET\Framework\v4.0.30319\Config folder).
2. Set the PrintingPermission level to NoPrinting.

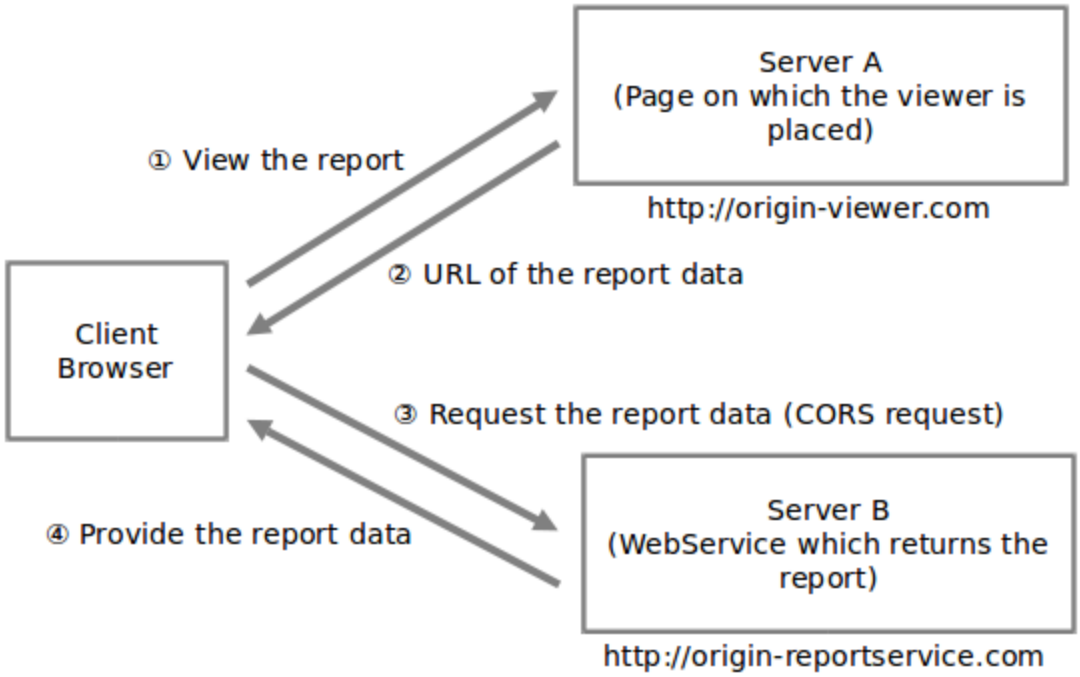
XML code. Paste BETWEEN the system.web tags.

```
<IPermission class="PrintingPermission"version="1"Level="NoPrinting"/>
```

Note: The default set of medium trust permissions is available in the web_mediumtrust.config.default file.

Viewing Reports from Different Domains using CORS

Cross-Origin Resource Sharing is a technology for the web that provides async web operations to directly access reports from different domains. CORS works by adding a special header to responses from a server to the client. If a response contains the Access-Control-Allow-Origin header, then you can directly access the reports from another domain.



The following viewers require CORS:

1. JSViewer
2. HTMLViewer (WebViewer control)
3. RawHTML (WebViewer control)

The following steps describe how to access reports from different domain using CORS.

1. Add Global Application Class file (Global.asax) in your application on the service side.
2. Open the Global.asax.cs file, and add the following code to access reports using cross websites.

Paste inside the Global.asax.cs

```
protected void Application_BeginRequest(object sender, EventArgs e)
{
  HttpContext.Current.Response.AddHeader("Access-Control-Allow-Origin", "*");
  if (HttpContext.Current.Request.HttpMethod == "OPTIONS")
    {
      HttpContext.Current.Response.AddHeader("Access-Control-Allow-Methods",
"GET, POST, OPTIONS");
      HttpContext.Current.Response.AddHeader("Access-Control-Allow-Headers",
"Content-Type, Accept");
      HttpContext.Current.Response.End();
    }
 }
```

Paste inside the Global.asax.vb

```
Sub Application_BeginRequest(ByVal sender As Object, ByVal e As EventArgs)
   HttpContext.Current.Response.AddHeader("Access-Control-Allow-Origin", "*")
   If HttpContext.Current.Request.HttpMethod = "OPTIONS" Then
```

```
        HttpContext.Current.Response.AddHeader("Access-Control-Allow-Methods",
 "GET, POST, OPTIONS")
        HttpContext.Current.Response.AddHeader("Access-Control-Allow-Headers",
 "Content-Type, Accept")
        HttpContext.Current.Response.End()
    End If
End Sub
```

Note:

- If you get error 404 or 500 on the report preview, please make sure that your browser supports CORS.

Print Reports

Learn to perform common tasks with ActiveReports with quick how-to topics.

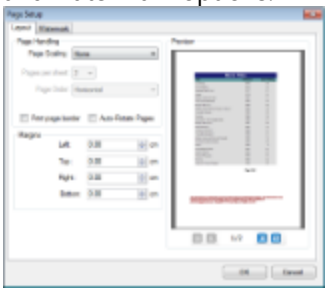
Topic	Content
Advanced Print Options	Learn how to access and set up the advanced printing options provided with the ActiveReports Viewer.
Print Methods	Learn about various Print methods in ActiveReports.
PDF Print Presets	Learn how to preset basic print options for PDF printing.
Print in JSViewer	Learn various methods of printing in JSViewer.

Advanced Print Options

The advanced printing options in the ActiveReports Viewer, allow you to change page scaling, set page margins and add a watermark when printing a report.

To access the advanced printing options

1. In the Viewer toolbar, click the **Print** button. See Windows Forms Viewer for information on the Viewer toolbar.
2. In the Print dialog that appears, click **Advanced**.
3. In the Page Setup dialog that appears, go to the **Layout** and **Watermark** tabs to set page scaling, page margins and watermark options.



To modify page scaling

1. In the Page Setup dialog, on the **Layout** tab under the Page Handling group, select a value from the **Page Scaling** drop-down list.
2. In case you select **Multiple pages per sheet** under Page Scaling, you can specify the **Pages per sheet** and **Page Order** options.
3. Click **OK** to close the dialog.

> **Note:** You may also check the **Print page border** or **Auto-Rotate Pages** check box for further customization of your print setup.

To change page margins

1. In the Page Setup dialog, on the **Layout** tab under the Margins group, enter values for the Left, Top, Right and Bottom margins.
2. Click **OK** to close the dialog.

To add a watermark to the report

1. In the Page Setup dialog, on the **Watermark** tab, under the textbox named **Text**, enter the text you want to display in your watermark.
2. Select the **Font**, **Size**, and **Color** for the text.
3. In the **Angle** field, enter a numeric value between 0 and 360 (A value of 0 renders straight left-to-right text. A value of 180 renders inverted text).
4. Click **OK** to close the dialog.

Print Methods

ActiveReports provides access to Print methods to enable printing of page and section reports. You can access Print methods in any of the following ways:

- **Viewer.Print method (using the Viewer control)**
- **Print methods in SectionDocument or PageDocument**
- **Print methods in the PrintExtension class**

Viewer.Print method

The code sample illustrates how to access the print method using the Viewer control.

You can use the **Print ('Print Method' in the on-line documentation)** method of the Viewer class to print a report loaded in the Viewer control. Make sure that the report is loaded completely before Print is executed.

Visual Basic.NET code. Add this code INSIDE the LoadCompleted event of the Viewer

```
Viewer1.Print(True, True, True)
```

C# code. Add this code INSIDE the LoadCompleted event of the Viewer

```
viewer1.Print(true, true, true);
```

Print methods in SectionDocument or PageDocument

SectionDocument and PageDocument types have Print methods that can be used directly on the document object. The following code samples illustrate how to access the print methods that can be used directly on the document object.

> **Note**: The **Print ('Print Method' in the on-line documentation)** method is implemented as an extension method of the **PrintExtension.Print ('Print Method' in the on-line documentation)** method, which is present in the GrapeCity.ActiveReports namespace of GrapeCity.ActiveReports.Viewer.Win assembly.
>
> In order to access **Print ('Print Method' in the on-line documentation)** method through SectionDocument or PageDocument class, you need to add GrapeCity.ActiveReports.Viewer.Win reference to the project. Also, as mentioned in the code, make sure that you add a reference for the GrapeCity.ActiveReports namespace in your project using Imports (Visual Basic.NET) or using (C#) statement.

Section Report

Visual Basic.NET code. Paste at the top of the code view.

```
Imports GrapeCity.ActiveReports
```

Visual Basic.NET code. Paste INSIDE the Form_Load event.

```
Dim rpt = New SectionReport1()
rpt.Run(False)
Dim sectionDocument = rpt.Document
sectionDocument.Print(True, True, False)
```

C# code. Paste at the top of the code view.

```
using GrapeCity.ActiveReports;
```

C# code. Paste INSIDE the Form_Load event.

```
var rpt = new SectionReport1();
rpt.Run(false);
var sectionDocument = rpt.Document;
sectionDocument.Print(true, true, false);
```

Page Report

Visual Basic.NET code. Paste at the top of the code view.

```
Imports GrapeCity.ActiveReports
```

Visual Basic.NET code. Paste INSIDE the Form_Load event.

```
Dim file_name As String = "..\..\PageReport1.rdlx"
Dim pageReport As New GrapeCity.ActiveReports.PageReport(New
System.IO.FileInfo(file_name))
Dim pageDocument As New GrapeCity.ActiveReports.Document.PageDocument(pageReport)
pageDocument.Print(True, True, False)
```

C# code. Paste at the top of the code view.

```
using GrapeCity.ActiveReports;
```

C# code. Paste INSIDE the Form_Load event.

```
string file_name = @"..\..\PageReport1.rdlx";
GrapeCity.ActiveReports.PageReport pageReport = new
GrapeCity.ActiveReports.PageReport(new System.IO.FileInfo(file_name));
GrapeCity.ActiveReports.Document.PageDocument pageDocument = new
GrapeCity.ActiveReports.Document.PageDocument(pageReport);
pageDocument.Print(true, true, false);
```

Print methods in the PrintExtension class

You can use the **Print ('Print Method' in the on-line documentation)** method of the PrintExtension class to print a report loaded in the Viewer control. Make sure that the report is loaded completely before Print is executed. The following code samples illustrate how to access the print method of the PrintExtension class.

> **Note**: The **Print ('Print Method' in the on-line documentation)** method is implemented as an extension method of the **PrintExtension.Print ('Print Method' in the on-line documentation)** method, which is present in the GrapeCity.ActiveReports namespace of GrapeCity.ActiveReports.Viewer.Win assembly.
>
> In order to access **Print ('Print Method' in the on-line documentation)** method through SectionDocument or PageDocument class, you need to add GrapeCity.ActiveReports.Viewer.Win reference to the project. Also, as mentioned in the code, make sure that you add a reference for the GrapeCity.ActiveReports namespace in your project using Imports (Visual Basic.NET) or using (C#) statement.

Section Report

Visual Basic.NET code. Paste INSIDE an event like Button_Click.

```
GrapeCity.ActiveReports.PrintExtension.Print(sectionDocument, True, True)
```

C# code. Paste INSIDE an event like Button_Click.

```
GrapeCity.ActiveReports.PrintExtension.Print(sectionDocument, true, true);
```

Page Report

Visual Basic.NET code. Paste INSIDE an event like Button_Click.

```
GrapeCity.ActiveReports.PrintExtension.Print(pageDocument, True, True)
```

C# code. Paste INSIDE an event like Button_Click.

```
GrapeCity.ActiveReports.PrintExtension.Print(pageDocument, true, true);
```

Print in JSViewer

JSViewer provides several options for printing a report. This topic describes several ways in which a report can be printed in JSViewer.

Print with Preview

Print report when the report is completely loaded in the viewer, using **print()** method.

index.html

```
var viewer;
function loadViewer() {
    viewer = GrapeCity.ActiveReports.JSViewer.create({
        element: '#viewerContainer',
        reportID: 'RdlReport1.rdlx',
        documentLoaded: () => viewer.print()
    });
}
```

Print without Preview

Print report without previewing using 'global' **print()** method. This is same as the default button in JSViewer, but without showing report preview.

index.html

```
GrapeCity.ActiveReports.JSViewer.print({ reportID:'RdlReport1.rdlx'});
```

Preview Report and Print to PDF

Open the report and export it to PDF with **PrintOnOpen** parameter set to 'true'. In this case, the exported PDF opens in new window of the browser, and the print dialog is displayed.

index.html

```
var viewer;
function loadViewer() {
    viewer = GrapeCity.ActiveReports.JSViewer.create({
        element: '#viewerContainer',
        reportID: 'RdlReport1.rdlx',
        documentLoaded: () => viewer.export('Pdf', null, true, { PrintOnOpen: 'true' })
    });
}
```

Print to PDF

Export the report to PDF using 'global' **export()** method and enable the **PrintOnOpen** option. In this case, the report is not opened.

index.html

```
GrapeCity.ActiveReports.JSViewer.export({
    reportID: 'RdlReport1.rdlx', exportType: 'Pdf', settings: { PrintOnOpen: 'true' },
    callback: (args) => { window.open(args) }
})
```

Note:

- For Section Reports, use **OnlyForPrint** instead of **PrintOnOpen** (for backward compatibility). For Page and RDL reports (.rdlx), anyone of PrintOnOpen or OnlyForPrint can be used.
- Use latest versions of Chrome, Firefox, and Chrome-based Edge for the described print features to work correctly.

PDF Print Presets

To economize your efforts each time a PDF document is printed, you can preset basic print options when exporting a report to a PDF format.

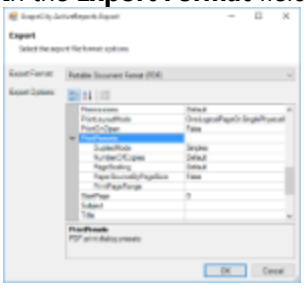
Note: The print preset properties are only available with the Professional Edition license. An evaluation message is displayed when used with the Standard Edition license.

In both Page/RDL and Section reports, you can set the PDF Print Preset properties using the **Export** dialog or through the code. The PDF print preset properties are available in the Export dialog of the following viewers.

- Standalone Designer
- End-User Designer
- WebViewer
- WPF Viewer

To set PDF print presets using the Export dialog

1. Open the **Export** dialog.
2. In the **Export Format** field of the **Export** dialog, select Portable Document Format (PDF).



3. Expand **PrintPresets** options and set the required properties for print presets.

   Note: These properties are available in PDF version 1.7 or higher. The PageScaling property is supported in PDF version 1.6.

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

To set PDF print presets through code

1. From the Visual Studio **File** menu, select **New**, then **Project**.
2. In the **New Project** dialog that appears, under language VB.NET or C#, click the **Reporting** node.
3. Select the type of report application that you want to add:
   - ActiveReports 14 Page Report Application
   - ActiveReports 14 RDL Report Application
   - ActiveReports 14 Section Report Application (xml-based)
4. In the **Name** field, enter a name for the report application, and click **OK**. The selected report type is added to your project.
5. In the Design view, double-click the Form title bar to create the Form_Load event.
6. Add the following code to invoke the Export methods and set print presets in the Form_Load event.

Section Report

Visual Basic.NET code. Paste INSIDE the Form_Load event

```
Dim sectionReport As New GrapeCity.ActiveReports.SectionReport()
Dim xtr As New System.Xml.XmlTextReader(Application.StartupPath +
"\..\..\SectionReport1.rpx")
sectionReport.LoadLayout(xtr)
sectionReport.Run()

'Define settings for PDF
Dim p As New GrapeCity.ActiveReports.Export.Pdf.Section.PdfExport()
p.Version = GrapeCity.ActiveReports.Export.Pdf.Section.PdfVersion.Pdf17

'Set default print settings using PrintPresets class
p.PrintPresets.PageScaling =
GrapeCity.ActiveReports.Export.Pdf.Enums.PageScaling.None
p.PrintPresets.DuplexMode =
GrapeCity.ActiveReports.Export.Pdf.Enums.DuplexMode.DuplexFlipLongEdge
p.PrintPresets.NumberOfCopies =
GrapeCity.ActiveReports.Export.Pdf.Enums.NumberOfCopies.Two
p.PrintPresets.PaperSourceByPageSize = True
p.PrintPresets.PrintPageRange = "1-3"
p.Export(sectionReport.Document, Application.StartupPath + "\PrintPresets.pdf")
```

C# code. Paste INSIDE the Form_Load event

```
GrapeCity.ActiveReports.SectionReport sectionReport = new
GrapeCity.ActiveReports.SectionReport();
System.Xml.XmlTextReader xtr = new System.Xml.XmlTextReader(Application.StartupPath
+ @"\..\..\SectionReport1.rpx");
sectionReport.LoadLayout(xtr);
```

```
sectionReport.Run();

//Define settings for PDF
GrapeCity.ActiveReports.Export.Pdf.Section.PdfExport p = new
GrapeCity.ActiveReports.Export.Pdf.Section.PdfExport();
p.Version = GrapeCity.ActiveReports.Export.Pdf.Section.PdfVersion.Pdf17;

//Set default print settings using PrintPresets class
p.PrintPresets.PageScaling =
GrapeCity.ActiveReports.Export.Pdf.Enums.PageScaling.None;
p.PrintPresets.DuplexMode =
GrapeCity.ActiveReports.Export.Pdf.Enums.DuplexMode.DuplexFlipLongEdge;
p.PrintPresets.NumberOfCopies =
GrapeCity.ActiveReports.Export.Pdf.Enums.NumberOfCopies.Two;
p.PrintPresets.PaperSourceByPageSize = true;
p.PrintPresets.PrintPageRange = "1-3";
p.Export(sectionReport.Document, Application.StartupPath + "\\PrintPresets.pdf");
```

Page/RDL Report

Visual Basic.NET code. Paste INSIDE the Form_Load event

```
'Set the rendering extension and render the report.
Dim pdfExport = New GrapeCity.ActiveReports.Export.Pdf.Page.PdfRenderingExtension()

'Define settings for PDF
Dim pdfSettings As New GrapeCity.ActiveReports.Export.Pdf.Page.Settings()
pdfSettings.Version = GrapeCity.ActiveReports.Export.Pdf.Page.PdfVersion.Pdf17
pdfSettings.PrintOnOpen = True

'Set default print settings using PrintPresets class
Dim pdfPresetsSetting As New GrapeCity.ActiveReports.Export.Pdf.PrintPresets()
pdfPresetsSetting.PageScaling =
GrapeCity.ActiveReports.Export.Pdf.Enums.PageScaling.None
pdfPresetsSetting.DuplexMode =
GrapeCity.ActiveReports.Export.Pdf.Enums.DuplexMode.DuplexFlipLongEdge
pdfPresetsSetting.NumberOfCopies =
GrapeCity.ActiveReports.Export.Pdf.Enums.NumberOfCopies.Two
pdfPresetsSetting.PaperSourceByPageSize = True
pdfPresetsSetting.PrintPageRange = "1-3"

pdfSettings.PrintPresets = pdfPresetsSetting

Dim outputFile = New IO.FileInfo("..\..\PrintPresets.pdf")
Dim reportFile = New IO.FileInfo("..\..\PageReport1.rdlx")

Dim fileStreamProvider = New
GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider(outputFile.Directory,
```

```
Path.GetFileNameWithoutExtension(outputFile.FullName))

Using pageDocument = New GrapeCity.ActiveReports.PageReport(reportFile).Document
    pageDocument.Render(pdfExport, fileStreamProvider, pdfSettings)
End Using
```

C# code. Paste INSIDE the Form_Load event

```
//Set the rendering extension and render the report.
var pdfExport = new
GrapeCity.ActiveReports.Export.Pdf.Page.PdfRenderingExtension();

//Define settings for PDF
GrapeCity.ActiveReports.Export.Pdf.Page.Settings pdfSettings = new
GrapeCity.ActiveReports.Export.Pdf.Page.Settings();
pdfSettings.Version = GrapeCity.ActiveReports.Export.Pdf.Page.PdfVersion.Pdf17;
pdfSettings.PrintOnOpen = true;

//Set default print settings using PrintPresets class
GrapeCity.ActiveReports.Export.Pdf.PrintPresets pdfPresetsSetting = new
GrapeCity.ActiveReports.Export.Pdf.PrintPresets();
pdfPresetsSetting.PageScaling =
GrapeCity.ActiveReports.Export.Pdf.Enums.PageScaling.None;
pdfPresetsSetting.DuplexMode =
GrapeCity.ActiveReports.Export.Pdf.Enums.DuplexMode.DuplexFlipLongEdge;
pdfPresetsSetting.NumberOfCopies =
GrapeCity.ActiveReports.Export.Pdf.Enums.NumberOfCopies.Two;
pdfPresetsSetting.PaperSourceByPageSize = true;
pdfPresetsSetting.PrintPageRange = "1-3";

pdfSettings.PrintPresets = pdfPresetsSetting;

var outputFile = new System.IO.FileInfo(@"..\..\PrintPresets.pdf");
var reportFile = new System.IO.FileInfo(@"..\..\PageReport1.rdlx");

var fileStreamProvider = new
GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider(outputFile.Directory,
System.IO.Path.GetFileNameWithoutExtension(outputFile.FullName));

using (var pageDocument = new
GrapeCity.ActiveReports.PageReport(reportFile).Document)
{
    pageDocument.Render(pdfExport, fileStreamProvider, pdfSettings);
}
```

Export Reports

In this section, learn about the independent ways to export a report to different formats, and some export

implementations.

Topic	Content
Rendering Extensions	Use the Render method in Rendering Extensions of the PageDocument class to render a page report and RDL Report to Image, Html, Pdf, Xml, Word and Excel formats.
Export Filters	Use the Export method of the corresponding ExportFilter class to export a section report, page report and RDL Reports. Exporting in Section Report is only possible through Export Filters

The following table illustrates the supported export formats for section, page and an RDL report.

Click the ✓ mark to see the implementation of the corresponding Export format.

Export formats		Section report	Page report	RDL report
HTML: Export reports to HTML or MHT formats, all of which open in a Web browser.	Rendering Extension	✘	✓	✓
	Export Filter	✓	✓	✓
Pdf: Export reports to PDF, a portable document format that opens in the Adobe Reader.	Rendering Extension	✘	✓	✓
	Export Filter	✓	✓	✓
Rtf: Export reports to RTF, RichText Format that opens in Microsoft Word, and is native to WordPad.		✓	✓	✓
Word: Export reports to DOC, a format that opens in Microsoft Word.	Word HTML (.doc)	✘	✓	✓
	LibreOffice (.docx)	✘	✓	✓
Text: Export reports to TXT, plain text format that opens in Notepad or any text editor. Export reports to CSV, comma separated values, a format that you can open in Microsoft Excel.		✓	✓	✓
Image: Export reports to BMP, GIF, JPEG, TIFF, and PNG image formats		✘	✓	✓
Tiff: Export reports to TIFF image format for optical archiving and faxing.		✓	✓	✓
Excel	Export Filter(XLS, XLSX)	✓	✓	✓
	Rendering Extension - Microsoft Excel Worksheet - Layout(XLS, XLSX)	✘	✓	✓
Xml: Export reports to XML, a format that opens in a Web browser or delivers data to other applications.		✘	✓	✓
CSV: Export reports to a CSV file, a form of structured data in plain text. The text in a CSV file is saved as series of values separated by comma.		✘	✓	✓
JSON: Export reports to a JSON file, a text-based data format in which the data is stored in the hierarchical form.		✘	✓	✓

Note: In **ASP.NET Core** applications, supported export formats are - Excel (.xlsx), Word (.docx), PDF, CSV, JSON, and TIFF.

Rendering Extensions

In ActiveReports, you can use the Render method in Rendering Extensions of the PageDocument class to render in any one of the following formats. Each exporting format provides it's own set of properties to control how the report is rendered.

- Rendering to HTML
- Rendering to PDF
- Rendering to Image
- Rendering to XML
- Rendering to Excel
- Rendering to Word
- Rendering to CSV
- Rendering to JSON
- Editable PDFs

Rendering to HTML

HTML, or hypertext markup language, is a format that opens in a Web browser. You can export your reports to HTML or MHT formats. It is a good format for delivering content because virtually all users have an HTML browser. The **HTMLRenderingExtension ('HtmlRenderingExtension Class' in the on-line documentation)** renders your report in this format with improved table border rendering and high quality SVG output for charts. If you do not want to use SVG in charts, set the RenderingEngine property to **Html**.

The following steps provide an example of rendering a report in the HTML format.

1. Create a new Visual Studio project.
2. In the **New Project** dialog that appears, select **ActiveReports 14 Page Report Application** and specify a name for the project in the Name field.
3. Click **OK** to create a new **ActiveReports 14 Page Report Application**. By default a Page report is added to the project.
4. Add reference to GrapeCity.ActiveReports.Export.Html package in the project.
5. On the Form.cs or Form.vb that opens, double-click the title bar to create the Form_Load event.
6. Add the following code inside the Form_Load event.

Visual Basic.NET code. Paste INSIDE the Form Load event.

```
' Provide the page report you want to render.
Dim report As New GrapeCity.ActiveReports.PageReport()
Dim reportDocument As New GrapeCity.ActiveReports.Document.PageDocument(report)

' Create an output directory.
Dim outputDirectory As New System.IO.DirectoryInfo("C:\MyHTML")
outputDirectory.Create()

' Provide settings for your rendering output.
Dim htmlSetting As New GrapeCity.ActiveReports.Export.Html.Page.Settings()
Dim setting As GrapeCity.ActiveReports.Extensibility.Rendering.ISettings =
htmlSetting
```

```
' Set the rendering extension and render the report.
Dim htmlRenderingExtension As New
GrapeCity.ActiveReports.Export.Html.Page.HtmlRenderingExtension()
Dim outputProvider As New
GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider(outputDirectory,
System.IO.Path.GetFileNameWithoutExtension(outputDirectory.Name))

' Overwrite output file if it already exists.
outputProvider.OverwriteOutputFile = True

reportDocument.Render(htmlRenderingExtension, outputProvider, htmlSetting)
```

C# code. Paste INSIDE the Form Load event.

```
// Provide the page report you want to render.
GrapeCity.ActiveReports.PageReport report = new
GrapeCity.ActiveReports.PageReport();
GrapeCity.ActiveReports.Document.PageDocument reportDocument = new
GrapeCity.ActiveReports.Document.PageDocument(report);

// Create an output directory.
System.IO.DirectoryInfo outputDirectory = new
System.IO.DirectoryInfo(@"C:\MyHTML");
outputDirectory.Create();

// Provide settings for your rendering output.
GrapeCity.ActiveReports.Export.Html.Page.Settings htmlSetting = new
GrapeCity.ActiveReports.Export.Html.Page.Settings();
GrapeCity.ActiveReports.Extensibility.Rendering.ISettings setting = htmlSetting;

// Set the rendering extension and render the report.
GrapeCity.ActiveReports.Export.Html.Page.HtmlRenderingExtension
htmlRenderingExtension = new
GrapeCity.ActiveReports.Export.Html.Page.HtmlRenderingExtension();
GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider outputProvider = new
GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider(outputDirectory,
System.IO.Path.GetFileNameWithoutExtension(outputDirectory.Name));

// Overwrite output file if it already exists.
outputProvider.OverwriteOutputFile = true;

reportDocument.Render(htmlRenderingExtension, outputProvider, htmlSetting);
```

HTML Rendering Properties

ActiveReports offers several options to control how reports render to HTML.

Property	Description

Fragment ('Fragment Property' in the on-line documentation)	Determine whether or not the full html text will be returned or just the contents contained inside the body tag will be returned. True indicates only the content inside the body tag will be return; otherwise false. The default is false.
MhtOutput ('MhtOutput Property' in the on-line documentation)	Gets or sets whether or not the output should be in Mht format. True indicates the output should be in Mht format; otherwise false. The default is false.
RenderingEngine ('RenderingEngine Property' in the on-line documentation)	The RenderingEngine property is set to **Mixed** by default for improved quality output. The choices are Html or Mixed, where Mixed uses SVG to render charts.
StyleStream ('StyleStream Property' in the on-line documentation)	Set the StyleStream to True to create an external .css file containing style information from your report controls' style properties. If you prefer to have style information embedded in the HTML file, set the StyleStream property to False.
LinkTarget ('LinkTarget Property' in the on-line documentation)	Specify a link target to control whether drill down reports and other links open in a new window or reuse the current window. By default, no value is set and links open in the same window. A value of _blank opens the link in a new window, or you can specify a window using window_name. By default this value is not set.
Mode ('Mode Property' in the on-line documentation)	Galley mode renders the report in one HTML stream. Select Paginated mode to render each page as a section inside the HTML document.
OutputTOC ('OutputTOC Property' in the on-line documentation)	Indicates whether the report's existing TOC should be added in the output.

Limitations

- HTML is not the best format for printing. Use the PDF rendering extension instead.
- Diagonal lines, Page margins, Border/Line Style (like DashedDotDot, DashDot, WindowInset) are not supported.
- Checkbox color does not affect the color of the square.
- TextIndent property and FillCharacter property of the TableOfContents control's Level setting are not supported.

Interactivity

Reports rendered in HTML support a number of interactive features. Hyperlinks, Bookmarks and Drill through links can be rendered to HTML. However, Document Maps are not available in this format. For a drill down report, make sure that the data you want to display is expanded before rendering, otherwise it renders in the hidden state.

Rendering to PDF

Portable Document Format (PDF), is a format recommended for printing and for preserving formatting. You can use the

PDFRenderingExtension ('PdfRenderingExtension Class' in the on-line documentation) to render your report in this format. With the PDF rendering extension, you can use features such as font linking, digital signatures and end-user defined characters (EUDC). These features are only available in the Professional Edition of ActiveReports.

The following steps provide an example of rendering a report in PDF format.

1. Create a new Visual Studio project.
2. In the **New Project** dialog that appears, select **ActiveReports 14 Page Report Application** and specify a name for the project in the Name field.
3. Click **OK** to create a new **ActiveReports 14 Page Report Application**. By default a Page report is added to the project.
4. Add reference to GrapeCity.ActiveReports.Export.Pdf package in the project.
5. On the Form.cs or Form.vb that opens, double-click the title bar to create the Form_Load event.
6. Add the following code inside the Form_Load event.

Visual Basic.NET code. Paste INSIDE the Form Load event.

```
' Provide the page report you want to render.
Dim rptPath As New IO.FileInfo("..\..\PageReport1.rdlx")
Dim pageReport As New GrapeCity.ActiveReports.PageReport(rptPath)

' Create an output directory.
Dim outputDirectory As New System.IO.DirectoryInfo("C:\MyPDF")
outputDirectory.Create()

' Provide settings for your rendering output.
Dim pdfSetting As New GrapeCity.ActiveReports.Export.Pdf.Page.Settings()

' Set the rendering extension and render the report.
Dim pdfRenderingExtension As New
GrapeCity.ActiveReports.Export.Pdf.Page.PdfRenderingExtension()
Dim outputProvider As New
GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider(outputDirectory,
System.IO.Path.GetFileNameWithoutExtension(outputDirectory.Name))

' Overwrite output file if it already exists
outputProvider.OverwriteOutputFile = True

pageReport.Document.Render(pdfRenderingExtension, outputProvider, pdfSetting)
```

C# code. Paste INSIDE the Form Load event.

```
// Provide the page report you want to render.
System.IO.FileInfo rptPath = new System.IO.FileInfo(@"..\..\PageReport1.rdlx");
GrapeCity.ActiveReports.PageReport pageReport = new
GrapeCity.ActiveReports.PageReport(rptPath);

// Create an output directory.
System.IO.DirectoryInfo outputDirectory = new System.IO.DirectoryInfo(@"C:\MyPDF");
```

```
outputDirectory.Create();

// Provide settings for your rendering output.
GrapeCity.ActiveReports.Export.Pdf.Page.Settings pdfSetting = new
GrapeCity.ActiveReports.Export.Pdf.Page.Settings();

// Set the rendering extension and render the report.
GrapeCity.ActiveReports.Export.Pdf.Page.PdfRenderingExtension pdfRenderingExtension =
new GrapeCity.ActiveReports.Export.Pdf.Page.PdfRenderingExtension();
GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider outputProvider = new
GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider(outputDirectory,
System.IO.Path.GetFileNameWithoutExtension(outputDirectory.Name));

// Overwrite output file if it already exists
outputProvider.OverwriteOutputFile = true;

pageReport.Document.Render(pdfRenderingExtension, outputProvider, pdfSetting);
```

PDF Rendering Properties

ActiveReports offers a number of options to control how reports render to PDF.

Property	Description
Application ('Application Property' in the online documentation)	Set the value that appears for application in the Document Properties dialog of the PDF viewer application.
Author ('Author Property' in the online documentation)	Enter the name of the author to appear in the Document Properties dialog of the PDF viewer application.
CenterWindow ('CenterWindow Property' in the online documentation)	Set to True to position the document's window in the center of the screen.
DisplayMode ('DisplayMode Property' in the online documentation)	Specifies how the document is displayed when opened. FullScreen mode displays the document with no menu bar, window controls, or any other window visible.
DisplayTitle ('DisplayTitle Property' in the online documentation)	Set to True to display text you enter in the Title property. When set to False it displays the name of the PDF file.
DpiX ('DpiX Property' in the on-	Set the horizontal resolution of the rendered PDF file.

line documentation)	
DpiY ('DpiY Property' in the online documentation)	Set the vertical resolution of the rendered PDF file.
EmbedFonts ('EmbedFonts Property' in the online documentation)	Select how the fonts used in the report should be embedded in the PDF document. **Note:** By default, all fonts get embedded in the exported PDF document.
Encrypt ('Encrypt Property' in the online documentation)	Determines whether the document is encrypted or not. **Note**: If Encrypt is set to False, permissions and passwords have no effect.
EndPage ('EndPage Property' in the online documentation)	The last page of the report to render. The default value is the value for StartPage, that is, 0.
FallbackFonts ('FallbackFonts Property' in the online documentation)	Gets or sets a comma-delimited string of font families to locate missing glyphs from the original font.
FitWindow ('FitWindow Property' in the online documentation)	True to resize the document's window to fit the size of the first displayed page. Default value: false.
HideMenubar ('HideMenubar Property' in the online documentation)	True to hide the viewer application's menu bar when the document is active. Default value: false.
HideToolbar ('HideToolbar Property' in the online documentation)	True to hide the viewer application's toolbars when the document is active. Default value: false.
HideWindowUI ('HideWindowUI Property' in the online documentation)	True to hide user interface elements in the document's window (such as scroll bars and navigation controls), leaving only the document's contents displayed. Default value: false.
ImageInterpolation ('ImageInterpolation Property' in the online documentation)	Interpolation value of images. Allows to enable/disable image interpolation, when exporting the file to PDF.
Keywords ('Keywords Property' in the online documentation)	Keywords associated with the document.

MarginBottom ('MarginBottom Property' in the online documentation)	The bottom margin value, in inches, to set for the report. You must include an integer or decimal value followed by "in" (for example, 1in). This value overrides the report's original settings.
MarginLeft ('MarginLeft Property' in the online documentation)	The left margin value, in inches, to set for the report. You must include an integer or decimal value followed by "in" (for example, 1in). This value overrides the report's original settings.
MarginRight ('MarginRight Property' in the online documentation)	The right margin value, in inches, to set for the report. You must include an integer or decimal value followed by "in" (for example, 1in). This value overrides the report's original settings.
MarginTop ('MarginTop Property' in the online documentation)	The top margin value, in inches, to set for the report. You must include an integer or decimal value followed by "in" (for example, 1in). This value overrides the report's original settings.
OwnerPassword ('OwnerPassword Property' in the online documentation)	The owner password that can be entered in the reader that permits full access to the document regardless of the specified user permissions.
PageHeight ('PageHeight Property' in the online documentation)	The page height value, in inches, to set for the report. You must include an integer or decimal value followed by "in" (for example, 1in). This value overrides the report's original settings.
PageWidth ('PageWidth Property' in the online documentation)	The page width value, in inches, to set for the report. You must include an integer or decimal value followed by "in" (for example, 1in). This value overrides the report's original settings.
Permissions ('Permissions Property' in the online documentation)	Specifies the user permissions for the document. Permissions can be combined using a comma between values. In order to use **AllowFillin**, **AllowAccessibleReaders**, and **AllowAssembly** permissions, you must set the **Use128Bit** property to **True**.
PrintLayoutMode ('PrintLayoutMode Property' in the online documentation)	Specifies layout mode to be used for PDF document.
PrintOnOpen ('PrintOnOpen Property' in the online documentation)	Gets or sets the value indicating whether the document should be printed after open.
PrintPresets ('PrintPresets Property' in the on-	Gets or sets the PDF print preset dialog.

line documentation)	
SizeToFit ('SizeToFit Property' in the on-line documentation)	Determines whether PDF pages are fit to the selected paper size or not.
StartPage ('StartPage Property' in the on-line documentation)	The first page of the report to render. A value of 0 indicates that all pages are rendered.
Subject ('Subject Property' in the on-line documentation)	The subject of the document.
Title ('Title Property' in the on-line documentation)	The title of the document.
Use128Bit ('Use128Bit Property' in the on-line documentation)	True to use 128 bit encryption with full permissions capability. False to use 40 bit encryption with limited permissions
UserPassword ('UserPassword Property' in the on-line documentation)	The user password that can be entered in the reader. If this value is left empty, the user will not be prompted for a password, however the user will be restricted by the specified permissions.
WatermarkAngle ('WatermarkAngle Property' in the on-line documentation)	Specify the degree of angle for the watermark text on the PDF document. Valid values range from 0 to 359, where 0 is horizontal, left to right.
WatermarkColor ('WatermarkColor Property' in the on-line documentation)	Select a color for the watermark text on the PDF document. The default value for the watermark color is gray, but you can select any Web, System, or Custom color.
WatermarkFont ('WatermarkFont Property' in the on-line documentation)	Set the font to use for the watermark to any valid System.Drawing.Font.
WatermarkTitle ('WatermarkTitle Property' in the on-line documentation)	Enter text (i.e. CONFIDENTIAL) to use as the watermark on the PDF document.

PDF Print Presets Properties

ActiveReports allows you to preset the printing properties for PDF report exports using the **PrintPresets ('PrintPresets Class' in the on-line documentation)** class. This prepopulates the print settings in the Print dialog box. Please see Use PDF Printing Presets for more information.

Note: The print preset properties are only available with the Professional Edition license. An evaluation message is displayed when used with the Standard Edition license.

Property	Description
PageScaling ('PageScaling Property' in the on-line documentation)	Specify scaling for the printable area.You can select Default to shrink to the printable area, or you can select None for the actual size.
DuplexMode ('DuplexMode Property' in the on-line documentation)	Specify the duplex mode of the printer. For the best results with the duplex option, the selected printer should support duplex printing. You can choose from the following values, • Simplex: Prints on one side of the paper. This is the default value. • Duplex (Flip on long edge): Prints on both sides of the paper with paper flip on the long edge. • Duplex (Flip on short edge): Prints on both sides of the paper with paper flip on the short edge.
PaperSourceByPageSize ('PaperSourceByPageSize Property' in the on-line documentation)	Determines the output tray based on PDF page size, rather than page setting options. This option is useful when printing PDFs with multiple page sizes, where different sized output trays are available. By default, this option is set to False.
PrintPageRange ('PrintPageRange Property' in the on-line documentation)	Specify the range of page numbers as 1-3 or 1, 2, 3.
NumberOfCopies ('NumberOfCopies Property' in the on-line documentation)	Specify the number of copies to print. You can select any number of copies from 2 to 5, or select Default to specify a single copy.

Note: These properties are available in PDF version 1.7 or higher. The **PageScaling** property is supported in PDF version 1.6.

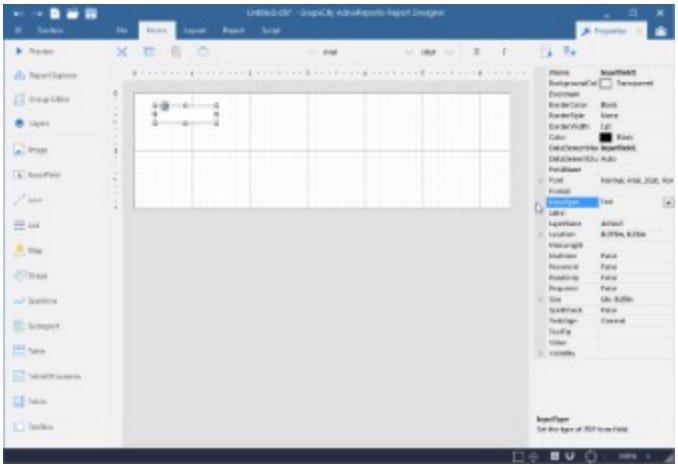
Interactivity

PDF is considered as the best format for printing and it also supports interactive features like Document Map, Bookmarks and Hyperlinks. However, in case you have any data hidden (like in a drill-down report) at the time of rendering, it does not show up in the output. Therefore, it is recommended to expand all toggle items prior to rendering.

Editable PDF: InputField Control

In Page and RDL reports, you can use the InputField report control. This control provides support for editable fields in an exported PDF report file where the InputField's value can be modified.

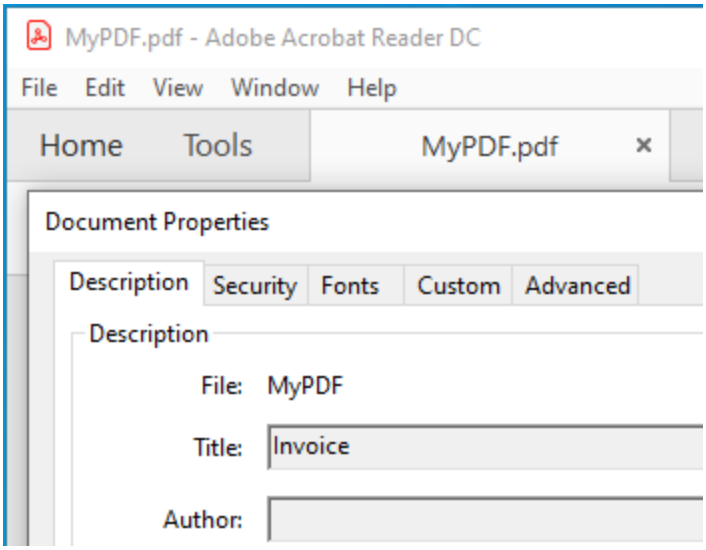
There are two (2) types of InputField - **Text** and **Checkbox**, which you can set in the **InputType** property. Each type has its own set of properties: the **Text** type of the InputField control gets the set of properties of the TextBox control. If the **Checkbox** type is selected, then the new control inherits the set of properties of the CheckBox control.



Note: The InputField control is part of the Professional Edition. With the Standard license, the InputField control is not displayed in an exported file.

Metadata in PDFs

Adding Metadata



Metadata such as keywords, descriptions are used by the search engines to narrow down the searches. You can add a number of predefined accessors, such as title, contributors, creators, copyright, description, etc using **AdditionalMetadata ('AdditionalMetadata Property' in the on-line documentation)** property. The allowed

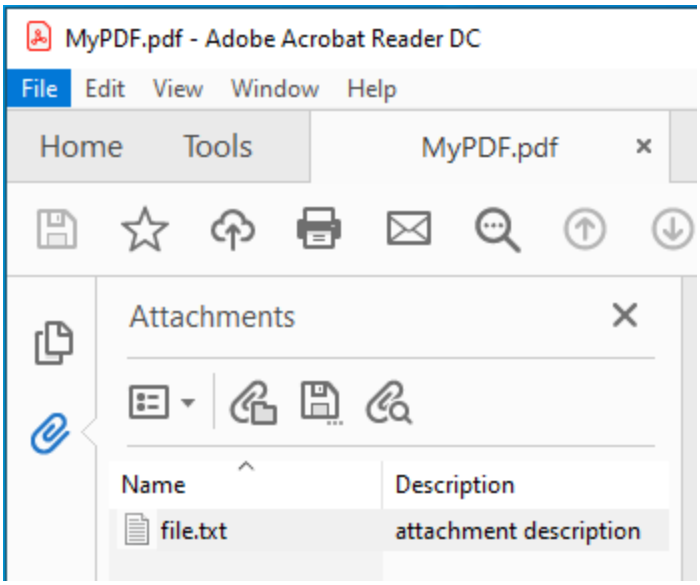
namespaces are:

- Dublin Core Properties
- XMP Core Properties
- PDF Properties

C# code. Paste INSIDE the Form Load event.

```
// Provide the page report you want to render.
System.IO.FileInfo rptPath = new System.IO.FileInfo(@"..\..\PageReport1.rdlx");
GrapeCity.ActiveReports.PageReport pageReport = new
GrapeCity.ActiveReports.PageReport(rptPath);
// Create an output directory.
System.IO.DirectoryInfo outputDirectory = new System.IO.DirectoryInfo(@"C:\MyPDF");
outputDirectory.Create();
// Add meta data.
var pdfSetting = new GrapeCity.ActiveReports.Export.Pdf.Page.Settings();
// using GrapeCity.ActiveReports.Export.Pdf;
pdfSetting.AdditionalMetadata.Add(new AdditionalMetadataInfo
{
    Namespace = AdditionalMetadataNamespace.PurlOrg, // Dublin Core Properties
    Key = "title",
    Value = "Invoice"
});
GrapeCity.ActiveReports.Export.Pdf.Page.PdfRenderingExtension pdfRenderingExtension =
new GrapeCity.ActiveReports.Export.Pdf.Page.PdfRenderingExtension();
GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider outputProvider = new
GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider(outputDirectory,
System.IO.Path.GetFileNameWithoutExtension(outputDirectory.Name));
outputProvider.OverwriteOutputFile = true;
pageReport.Document.Render(pdfRenderingExtension, outputProvider, pdfSetting);
```

Adding Attachment



You can include an attachment as metadata (such as invoices) to exported PDFs using **Attachments ('Attachments Property' in the on-line documentation)** property. This property allows to attach files such as a .xml or a .txt file in PDF. Below is the code example to export RDL and Page reports to PDF and attach a file to the exported PDF.

VB code. Paste INSIDE the Form Load event.

```
' Provide the page report you want to render.
Dim rptPath As System.IO.FileInfo = New System.IO.FileInfo("..\..\PageReport1.rdlx")
Dim pageReport As GrapeCity.ActiveReports.PageReport = New
GrapeCity.ActiveReports.PageReport(rptPath)
' Create an output directory.
Dim outputDirectory As System.IO.DirectoryInfo = New System.IO.DirectoryInfo("C:\MyPDF")
outputDirectory.Create()
' Add attachment.
Dim pdfSetting = New GrapeCity.ActiveReports.Export.Pdf.Page.Settings()
pdfSetting.Attachments.Add(New AttachmentInfo With {
   .Name = "file.txt",
   .Content = System.IO.File.ReadAllBytes("D:\Reports\file.txt"),
   .Description = "attachment description" ' optional
})
Dim pdfRenderingExtension As
GrapeCity.ActiveReports.Export.Pdf.Page.PdfRenderingExtension = New
GrapeCity.ActiveReports.Export.Pdf.Page.PdfRenderingExtension()
Dim outputProvider As GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider = New
GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider(outputDirectory,
System.IO.Path.GetFileNameWithoutExtension(outputDirectory.Name))
outputProvider.OverwriteOutputFile = True
pageReport.Document.Render(pdfRenderingExtension, outputProvider, pdfSetting)
```

C# code. Paste INSIDE the Form Load event.

```
// Provide the page report you want to render.
System.IO.FileInfo rptPath = new System.IO.FileInfo(@"..\..\PageReport1.rdlx");
```

```
GrapeCity.ActiveReports.PageReport pageReport = new
GrapeCity.ActiveReports.PageReport(rptPath);
// Create an output directory.
System.IO.DirectoryInfo outputDirectory = new System.IO.DirectoryInfo(@"C:\MyPDF");
outputDirectory.Create();
// Add attachment.
var pdfSetting = new GrapeCity.ActiveReports.Export.Pdf.Page.Settings();
// using GrapeCity.ActiveReports.Export.Pdf;
pdfSetting.Attachments.Add(new AttachmentInfo
{
    Name = "file.txt",
    Content = System.IO.File.ReadAllBytes(@"D:\Reports\file.txt"),
    Description = "attachment description" // optional
});
// or
//{
//   Name = "file.xml",
//   Content = File.ReadAllBytes(Application.StartupPath + "\\file.xml")
//};
GrapeCity.ActiveReports.Export.Pdf.Page.PdfRenderingExtension pdfRenderingExtension =
new GrapeCity.ActiveReports.Export.Pdf.Page.PdfRenderingExtension();
GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider outputProvider = new
GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider(outputDirectory,
System.IO.Path.GetFileNameWithoutExtension(outputDirectory.Name));
outputProvider.OverwriteOutputFile = true;
pageReport.Document.Render(pdfRenderingExtension, outputProvider, pdfSetting);
```

Open the exported PDF and you should see the attachment. Check the left sidebar in Adobe Acrobat Reader DC.

Note: Metadata in PDFs is part of the Professional Edition. It is supported with the PDF version PDF/A-3b (or higher).

PDF/A Support Limitations

- The **NeverEmbedFonts** property is ignored, so all fonts of a report are embedded into the PDF document.
- The **Security.Encrypt** property is ignored and the PDF export behaves as if this property is always set to False.
- The **OnlyForPrint** property is ignored and the PDF export behaves as if this property is always set to False.
- The **DocumentToAddBeforeReport** and **DocumentToAddAfterReport** properties of the PDF Rendering Extension settings are ignored.
- **Transparent images** lose their transparency when exported to PDF/A-1.
- **External hyperlinks** are exported as plain text.

Rendering to Images

Image is the format that converts your report to an image file. You can use the **ImageRenderingExtension ('ImageRenderingExtension Class' in the on-line documentation)** to your render you report in this format. Make sure that you select an **ImageType ('ImageType Property' in the on-line documentation)** to any of the six different image formats available: BMP, GIF, JPEG, TIFF, and PNG.

> **Note:** By default, the image rendering extension creates a separate file for each page in a report and adds an index to each corresponding file name (for example, image001.PNG, image002.PNG, etc).
>
> To render the entire report as a single image, set the **Pagination ('Pagination Property' in the on-line documentation)** setting to **False**.

The following steps provide an example of rendering a report in Image format.

1. Create a new Visual Studio project.
2. In the **New Project** dialog that appears, select **ActiveReports 14 Page Report Application** and specify a name for the project in the Name field.
3. Click **OK** to create a new **ActiveReports 14 Page Report Application**. By default a Page report is added to the project.
4. Add reference to GrapeCity.ActiveReports.Export.Image package in the project.
5. On the Form.cs or Form.vb that opens, double-click the title bar to create the Form_Load event.
6. Add the following code inside the Form_Load event.

Visual Basic.NET code. Paste INSIDE the Form Load event.

```
' Provide the page report you want to render.
Dim report As New GrapeCity.ActiveReports.PageReport()
Dim reportDocument As New GrapeCity.ActiveReports.Document.PageDocument(report)

' Create an output directory.
Dim outputDirectory As New System.IO.DirectoryInfo("C:\MyImage")
outputDirectory.Create()

' Provide settings for your rendering output.
Dim imageSetting As New GrapeCity.ActiveReports.Export.Image.Page.Settings()
Dim setting As GrapeCity.ActiveReports.Extensibility.Rendering.ISettings = imageSetting

' Set the rendering extension and render the report.
Dim imageRenderingExtension As New
GrapeCity.ActiveReports.Export.Image.Page.ImageRenderingExtension()
Dim outputProvider As New
GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider(outputDirectory,
System.IO.Path.GetFileNameWithoutExtension(outputDirectory.Name))

' Overwrite output file if it already exists.
outputProvider.OverwriteOutputFile = True

reportDocument.Render(imageRenderingExtension, outputProvider, imageSetting)
```

C# code. Paste INSIDE the Form Load event.

```
// Provide the page report you want to render.
GrapeCity.ActiveReports.PageReport report = new GrapeCity.ActiveReports.PageReport();
GrapeCity.ActiveReports.Document.PageDocument reportDocument = new
GrapeCity.ActiveReports.Document.PageDocument(report);
```

```
// Create an output directory.
System.IO.DirectoryInfo outputDirectory = new System.IO.DirectoryInfo(@"C:\MyImage");
outputDirectory.Create();

// Provide settings for your rendering output.
GrapeCity.ActiveReports.Export.Image.Page.Settings imageSetting = new
GrapeCity.ActiveReports.Export.Image.Page.Settings();
GrapeCity.ActiveReports.Extensibility.Rendering.ISettings setting = imageSetting;

// Set the rendering extension and render the report.
GrapeCity.ActiveReports.Export.Image.Page.ImageRenderingExtension
imageRenderingExtension = new
GrapeCity.ActiveReports.Export.Image.Page.ImageRenderingExtension();
GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider outputProvider = new
GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider(outputDirectory,
System.IO.Path.GetFileNameWithoutExtension(outputDirectory.Name));

// Overwrite output file if it already exists.
outputProvider.OverwriteOutputFile = true;

reportDocument.Render(imageRenderingExtension, outputProvider, imageSetting);
```

Image Rendering Properties

ActiveReports offers several options to control how reports render to Image.

Property	Description
Compression ('Compression Property' in the on-line documentation)	Sets or returns a value which specifies the compression to be used when exporting.
Dither ('Dither Property' in the on-line documentation)	Specifies whether the image should be dithered when saving to a black and white output format, like CCITT3 or Rle. This property has no effect if the CompressionScheme property is set to Lzw or None(represents color output).
DpiX ('DpiX Property' in the on-line documentation)	Adjust the horizontal resolution of rendered images. The default value is 96.
DpiY ('DpiY Property' in the on-line documentation)	Adjust the vertical resolution of rendered images.
EndPage ('EndPage Property' in the on-line documentation)	The default value of **0** in this property renders all of the report pages. Otherwise, set this value to the number of the last page to render. Please note that if the **StartPage** property is set to **0**, all of the pages of the report render. In order to use the **EndPage** property, you must set the **StartPage** property to a valid non-zero number.

ImageType ('ImageType Property' in the online documentation)	Select the type of image to which you want to render the report. Supported types are BMP, GIF, JPEG, TIFF, and PNG.
MarginBottom ('MarginBottom Property' in the online documentation)	Set the value in inches to use for the bottom margin of the image. The format is an integer or decimal with "in" as the suffix, for example "1in" for 1 inch. By default, no margins are used. The value set in this property overrides the report's settings.
MarginLeft ('MarginLeft Property' in the online documentation)	Set the value in inches to use for the left margin of the image. The format is an integer or decimal with "in" as the suffix, for example "1in" for 1 inch. By default, no margins are used. The value set in this property overrides the report's settings.
MarginRight ('MarginRight Property' in the online documentation)	Set the value in inches to use for the right margin of the image. The format is an integer or decimal with "in" as the suffix, for example "1in" for 1 inch. By default, no margins are used. The value set in this property overrides the report's settings.
MarginTop ('MarginTop Property' in the online documentation)	Set the value in inches to use for the top margin of the image. The format is an integer or decimal with "in" as the suffix, for example "1in" for 1 inch. By default, no margins are used. The value set in this property overrides the report's settings.
PageHeight ('PageHeight Property' in the online documentation)	Set the value in inches to use for the height of the image. The format is an integer or decimal with "in" as the suffix, for example "11in" for 11 inches. The value set in this property overrides the report's settings.
PageWidth ('PageWidth Property' in the online documentation)	Set the value in inches to use for the height of the image. The format is an integer or decimal with "in" as the suffix, for example "11in" for 11 inches. The value set in this property overrides the report's settings.
Pagination ('Pagination Property' in the online documentation)	By default, each page of a report is rendered as a separate image. Set this value to **False** to render the entire report as a single image.
PrintLayoutMode ('PrintLayoutMode Property' in the online documentation)	Select how to lay out the pages of the report in the image. • **OneLogicalPageOnSinglePhysicalPage** • **TwoLogicalPagesOnSinglePhysicalPage** • **FourLogicalPagesOnSinglePhysicalPage** • **EightLogicalPagesOnSinglePhysicalPage** • **BookletMode** (lays the pages out for booklet printing)
Quality ('Quality Property' in the online documentation)	Gets or sets the quality of the report to be rendered as an image.

SizeToFit ('SizeToFit Property' in the on-line documentation)	By default, rendered report pages are not resized to fit within the selected image size. Set this value to **True** to resize the report pages.
Start Page ('StartPage Property' in the on-line documentation)	The default value of zero in this and the **EndPage** properties render all of the report pages to images. Otherwise, set this value to the number of the first page to render.
WatermarkAngle ('WatermarkAngle Property' in the on-line documentation)	Specify the degree of angle for the watermark text on the image. Valid values range from 0 to 359, where 0 is horizontal, left to right.
WatermarkColor ('WatermarkColor Property' in the on-line documentation)	Select a color for the watermark text on the image. The default value for the watermark color is gray, but you can select any Web, System, or Custom color.
WatermarkFont ('WatermarkFont Property' in the on-line documentation)	Set the font to use for the watermark to any valid System.Drawing.Font.
WatermarkTitle ('WatermarkTitle Property' in the on-line documentation)	Sets text (i.e. CONFIDENTIAL) to use as the watermark on the image.

Interactivity

Reports rendered as images do not support any of the interactive features of Data Dynamics Reports. Any data hidden at the time of export is hidden in the image. To display all data in a drill-down report, expand all toggle items prior to exporting.

Rendering to XML

XML is a useful format for delivering data to other applications as the resulting XML file opens in an internet browser. You can use the **XmlRenderingExtension ('XmlRenderingExtension Class' in the on-line documentation)** to render your report in this format. XML is a good format for delivering data to other applications. The resulting XML file opens in an internet browser

The following steps provide an example of rendering a report in the Xml format.

1. Create a new Visual Studio project.
2. In the **New Project** dialog that appears, select **ActiveReports 14 Page Report Application** and specify a name for the project in the Name field.
3. Click **OK** to create a new **ActiveReports 14 Page Report Application**. By default a Page report is added to the project.
4. Add reference to GrapeCity.ActiveReports.Export.Xml package in the project.

5. On the Form.cs or Form.vb that opens, double-click the title bar to create the Form_Load event.
6. Add the following code inside the Form_Load event.

Visual Basic.NET code. Paste INSIDE the Form Load event.

```
' Provide the page report you want to render.
Dim report As New GrapeCity.ActiveReports.PageReport()Dim reportDocument As New
GrapeCity.ActiveReports.Document.PageDocument(report)

' Create an output directory.
Dim outputDirectory As New System.IO.DirectoryInfo("C:\MyXml")
outputDirectory.Create()

' Provide settings for your rendering output.
Dim xmlSetting As New GrapeCity.ActiveReports.Export.Xml.Page.Settings()
Dim setting As GrapeCity.ActiveReports.Extensibility.Rendering.ISettings = xmlSetting

' Set the rendering extension and render the report.
Dim xmlRenderingExtension As New
GrapeCity.ActiveReports.Export.Xml.Page.XmlRenderingExtension()Dim outputProvider As New
GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider(outputDirectory,
System.IO.Path.GetFileNameWithoutExtension(outputDirectory.Name))

' Overwrite output file if it already exists.
outputProvider.OverwriteOutputFile = True

reportDocument.Render(xmlRenderingExtension, outputProvider, xmlSetting)
```

C# code. Paste INSIDE the Form Load event.

```
// Provide the page report you want to render.
GrapeCity.ActiveReports.PageReport report = new
GrapeCity.ActiveReports.PageReport();GrapeCity.ActiveReports.Document.PageDocument
reportDocument = new GrapeCity.ActiveReports.Document.PageDocument(report);

// Create an output directory.
System.IO.DirectoryInfo outputDirectory = new System.IO.DirectoryInfo(@"C:\MyXml");
outputDirectory.Create();

// Provide settings for your rendering output.
GrapeCity.ActiveReports.Export.Xml.Page.Settings xmlSetting = new
GrapeCity.ActiveReports.Export.Xml.Page.Settings();
GrapeCity.ActiveReports.Extensibility.Rendering.ISettings setting = xmlSetting;

// Set the rendering extension and render the report.
GrapeCity.ActiveReports.Export.Xml.Page.XmlRenderingExtension xmlRenderingExtension =
new GrapeCity.ActiveReports.Export.Xml.Page.XmlRenderingExtension();
GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider outputProvider = new
GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider(outputDirectory,
```

```
System.IO.Path.GetFileNameWithoutExtension(outputDirectory.Name));

// Overwrite output file if it already exists.
outputProvider.OverwriteOutputFile = true;

reportDocument.Render(xmlRenderingExtension, outputProvider, xmlSetting);
```

Xml Rendering Properties

ActiveReports offers several options to control how reports render to Xml.

Property	Description
Encoding ('Encoding Property' in the on-line documentation)	Select the encoding schema to use in the XML transformation.
XslStylesheet ('XslStylesheet Property' in the on-line documentation)	Select the existing XSL Stylesheet file to use to transform the resulting XML file. **Note**: When using the XslStylesheet option, be sure to save the file in the correct file format, such as HTML.

Controlling XML Output

You can also control XML output through properties on the individual report controls. These properties are:

- **DataElementName** Indicates the name to use for the data element or attribute.
- **DataElementOutput** Indicates whether the report controls renders in the XML output.
- **DataElementStyle** Indicates whether a text box renders as an element or an attribute.
- **DetailDataCollectionName** Indicates the name to use for the collection of all instances of the detail group in the XML output.
- **DetailDataElementName** Indicates the name to use for instances of the detail group in the XML output.
- **DetailDataElementOutput** Indicates whether the details appear in the XML output.
- **DataInstanceElementOutput** Indicates whether a list appears in the XML output. (This property is ignored if there is a grouping in the list.)
- **DataInstanceName** Indicates the name to use for instances of the list in the XML output.

Interactivity

XML format does not support interactive features except that when rendering a report to XML, complete drill-down data is shown regardless of whether the data is rendered in expanded state or not.

Rendering to Excel

Microsoft Excel is one of the formats to which you can render your report using **ExcelRenderingExtension ('ExcelRenderingExtension Class' in the on-line documentation)**. You can export excel files in two formats, i.e. Xls and Xlsx.

The following steps provide an example of rendering a report in the Microsoft Excel format.

1. Create a new Visual Studio project.
2. In the **New Project** dialog that appears, select **ActiveReports 14 Page Report Application** and specify a name for the project in the Name field.
3. Click **OK** to create a new **ActiveReports 14 Page Report Application**. By default a Page report is added to the project.
4. Add reference to GrapeCity.ActiveReports.Export.Excel package in the project.
5. On the Form.cs or Form.vb that opens, double-click the title bar to create the Form_Load event.
6. Add the following code inside the Form_Load event.

Visual Basic.NET code. Paste INSIDE the Form Load event.

```
' Provide the page report you want to render.
Dim report As New GrapeCity.ActiveReports.PageReport()
Dim reportDocument As New GrapeCity.ActiveReports.Document.PageDocument(report)

' Create an output directory.
Dim outputDirectory As New System.IO.DirectoryInfo("C:\MyExcel")
outputDirectory.Create()

' Provide settings for your rendering output.
Dim excelSetting As New
GrapeCity.ActiveReports.Export.Excel.Page.ExcelRenderingExtensionSettings()
excelSetting.FileFormat = GrapeCity.ActiveReports.Export.Excel.Page.FileFormat.Xls
Dim setting As GrapeCity.ActiveReports.Extensibility.Rendering.ISettings =
excelSetting

' Set the rendering extension and render the report.
Dim excelRenderingExtension As New
GrapeCity.ActiveReports.Export.Excel.Page.ExcelRenderingExtension()
Dim outputProvider As New
GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider(outputDirectory,
System.IO.Path.GetFileNameWithoutExtension(outputDirectory.Name))

' Overwrite output file if it already exists.
outputProvider.OverwriteOutputFile = True

reportDocument.Render(excelRenderingExtension, outputProvider,
setting.GetSettings())
```

C# code. Paste INSIDE the Form Load event.

```
// Provide the page report you want to render.
GrapeCity.ActiveReports.PageReport report = new
```

```
GrapeCity.ActiveReports.PageReport();
GrapeCity.ActiveReports.Document.PageDocument reportDocument = new
GrapeCity.ActiveReports.Document.PageDocument(report);

// Create an output directory.
System.IO.DirectoryInfo outputDirectory = new
System.IO.DirectoryInfo(@"C:\MyExcel");
outputDirectory.Create();

// Provide settings for your rendering output.
GrapeCity.ActiveReports.Export.Excel.Page.ExcelRenderingExtensionSettings
excelSetting = new
GrapeCity.ActiveReports.Export.Excel.Page.ExcelRenderingExtensionSettings();
excelSetting.FileFormat = GrapeCity.ActiveReports.Export.Excel.Page.FileFormat.Xls;
GrapeCity.ActiveReports.Extensibility.Rendering.ISettings setting = excelSetting;

// Set the rendering extension and render the report.
GrapeCity.ActiveReports.Export.Excel.Page.ExcelRenderingExtension
excelRenderingExtension = new
GrapeCity.ActiveReports.Export.Excel.Page.ExcelRenderingExtension();
GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider outputProvider = new
GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider(outputDirectory,
System.IO.Path.GetFileNameWithoutExtension(outputDirectory.Name));

// Overwrite output file if it already exists.
outputProvider.OverwriteOutputFile = true;

reportDocument.Render(excelRenderingExtension, outputProvider,
setting.GetSettings());
```

Excel Rendering Properties

ActiveReports offers several options to control how reports render to Microsoft Excel.

Property	Description
PageSettings ('PageSettings Property' in the on-line documentation)	Returns an **ExcelRenderingExtensionPageSettings ('ExcelRenderingExtensionPageSettings Class' in the on-line documentation)** object for initializing Excel file print setting.
Pagination ('Pagination Property' in the on-line documentation)	Forces pagination or galley report layout mode.
RightToLeft	Shows direction of sheets from right to left.

('RightToLeft Property' in the on-line documentation)	
Security ('Security Property' in the on-line documentation)	Returns an **ExcelRenderingExtensionSecurity ('ExcelRenderingExtensionSecurity Class' in the on-line documentation)** object for initializing document security.
UseDefaultPalette ('UseDefaultPalette Property' in the on-line documentation)	Indicates whether to export the document with the default Excel palette.
FileFormat ('FileFormat Property' in the on-line documentation)	Specifies the output format of the Excel document, i.e. Xls or Xlsx.
OpenXmlStandard ('OpenXmlStandard Property' in the on-line documentation)	Specifies the level of Open XML document conformance on exporting in Xlsx file format. You can choose from the following values: **Strict** The default value. **Transitional** The Excel file generated by scheduled task execution using Strict (the default value of OpenXMLStandard) cannot be viewed on IOS devices. To change the default value of OpenXMLStandard, add following configuration settings in GrapeCity.ActiveReports.config file: <ReportingConfiguration> <Extensions> <Render> <Extension Name="Excel" Type="GrapeCity.ActiveReports.Export.Excel.Page.ExcelRenderingExtension, GrapeCity.ActiveReports.Export.Excel, Culture=neutral, PublicKeyToken=cc4967777c49a3ff" > <Settings> <Property Name="OpenXmlStandard" Value="Transitional" Type="GrapeCity.ActiveReports.Export.Excel.Page.OpenXmlStandard, GrapeCity.ActiveReports.Export.Excel, Culture=neutral, PublicKeyToken=cc4967777c49a3ff"/> </Settings> </Extension> </Render> </Extensions> </ReportingConfiguration>
MultiSheet	Indicates whether to generate a single-sheet or multi-sheet Excel document.

('MultiSheet Property' in the on-line documentation)	
EnableToggles ('EnableToggles Property' in the on-line documentation)	Allows to export collapsible rows in the detail and row groups of the **Table** control of an RDL report. This property gets displayed in the Export menu when the **Pagination** property is set to **False**.

Interactivity

Reports rendered in Excel support a number of interactive features like Bookmarks and Hyperlinks. However, in case you have any data hidden at the time of rendering (like in a drill-down report), it does not show up in the output. It is recommended that you expand all toggle items prior to rendering.

Limitations

- BackgroundImage and rounded corners (for Shape and Container) are not exported.
- Overlapping controls are not supported and an incorrect result will be obtained in the case of export.
- LineSpacing is not retained after export.
- Exported FormattedText control does not preserve styles and formatting. It exports FormattedText as TextBox with plain text without any tags.
- Barcodes are exported as an image object so scanning of barcode image may fail in some cases. It depends on printer settings and the scanner quality. Barcodes may be blurred on export and the caption may get truncated in case of physical printing.
- Exported Boolean values are displayed in uppercase in both Xls and Xlsx files.
- TextIndent property and FillCharacter property of the TableOfContents control's Level setting are not supported.
- Text decoration (Underline and LineThrough) gets applied to the indented area, when left padding is applied to the TableOfControl's levels.
- When a report is exported to Excel, CharWrap mode is ignored.
- Following properties of ActiveReports are not exported to Excel, since Excel does not have such settings:
    - CharacterSpacing
    - BorderWidth
    - MinCondenseRate

Rendering to Word

The **WordRenderingExtension ('WordRenderingExtension Class' in the on-line documentation)** class renders your reports to the native Microsoft Word file formats. You can export Page reports and RDL reports to **Microsoft Office Open XML (OOXML)** format (.Docx) or Word HTML format (.Doc) using the **FileFormat ('FileFormat Property' in the on-line documentation)** property.

The Word HTML format (.Doc) provides greater layout accuracy for Page and RDL Reports in Microsoft Word, on the other hand, OOXML format (.Docx) provides excellent editing experience for the exported reports.

The OOXML format (.Docx) is recommended in the following scenarios:

- **Open exported reports in a wide range of applications:** Users can open and modify the exported Word document in any of the following applications.
  - Microsoft Office 2007 - 2013
  - Microsoft Office for Mac 2008 - 2011
  - iWord and Pages v4 or higher for OS X
  - LibreOffice
  - Google Quickoffice for Android
  - Documents Free (Mobile Office Suite) by SavySoda for iOS

  > **Note:** Besides the applications listed above, .Docx files exported through ActiveReports WordRenderingExtension class might work in other applications that support the .Docx format.
- **Customize reports after exporting:** Positioning and arrangement of report elements in the exported document is implemented using the OOXML format (.Docx) which provides a natural document flow for editing the exported documents.
- **Use Word automation features:** With support for automation features in the OOXML format (.Docx), tasks that previously required manual adjustments in the exported Word document are now handled automatically. Report elements such as page header and footer, expressions, heading levels, and table of contents are automatically transformed to the OOXML format (.Docx).
- **Set compatibility mode:** You can render a report as a Word document that is compatible with Microsoft Word 2007-2010 as well as Microsoft Word 2013 using the DocumentCompatibleVersion property.

The following steps provide an example of rendering a report in the Word format (.doc or .docx).

1. Create a new Visual Studio project.
2. In the **New Project** dialog that appears, select **ActiveReports 14 Page Report Application** and specify a name for the project in the Name field.
3. Click **OK** to create a new **ActiveReports 14 Page Report Application**. By default a Page report is added to the project.
4. Add reference to GrapeCity.ActiveReports.Export.Word package in the project.
5. On the Form.cs or Form.vb that opens, double-click the title bar to create the Form_Load event.
6. Add the following code inside the Form_Load event to render your report in .OOXML or .HTML file format.

> **Note:** To export your report in Word HTML format (.Doc), change the **FileFormat** property option from OOXML to HTML format as depicted below.
> `wordSetting.FileFormat = GrapeCity.ActiveReports.Export.Word.Page.FileFormat.HTML`

To export a report in .Docx file format

Visual Basic.NET code. Paste INSIDE the Form Load event.

```
' Provide the page report you want to render.
Dim report As New GrapeCity.ActiveReports.PageReport()
Dim reportDocument As New GrapeCity.ActiveReports.Document.PageDocument(report)

' Create an output directory.
Dim outputDirectory As New System.IO.DirectoryInfo("C:\MyWord")
outputDirectory.Create()

' Provide settings for your rendering output.
Dim wordSetting As New GrapeCity.ActiveReports.Export.Word.Page.Settings()
```

```
' Set the FileFormat property to .OOXML.
wordSetting.FileFormat = GrapeCity.ActiveReports.Export.Word.Page.FileFormat.OOXML

' Set the rendering extension and render the report.
Dim wordRenderingExtension As New
GrapeCity.ActiveReports.Export.Word.Page.WordRenderingExtension()
Dim outputProvider As New
GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider(outputDirectory,
System.IO.Path.GetFileNameWithoutExtension(outputDirectory.Name))

' Overwrite output file if it already exists.
outputProvider.OverwriteOutputFile = True

reportDocument.Render(wordRenderingExtension, outputProvider, wordSetting)
```

C# code. Paste INSIDE the Form Load event.

```
// Provide the page report you want to render.
GrapeCity.ActiveReports.PageReport report = new GrapeCity.ActiveReports.PageReport();
GrapeCity.ActiveReports.Document.PageDocument reportDocument = new
GrapeCity.ActiveReports.Document.PageDocument(report);

// Create an output directory.
System.IO.DirectoryInfo outputDirectory = new System.IO.DirectoryInfo(@"C:\MyWord");
outputDirectory.Create();

// Provide settings for your rendering output.
GrapeCity.ActiveReports.Export.Word.Page.Settings wordSetting = new
GrapeCity.ActiveReports.Export.Word.Page.Settings();

// Set the FileFormat property to .OOXML.
wordSetting.FileFormat = GrapeCity.ActiveReports.Export.Word.Page.FileFormat.OOXML;

// Set the rendering extension and render the report.
GrapeCity.ActiveReports.Export.Word.Page.WordRenderingExtension wordRenderingExtension =
new GrapeCity.ActiveReports.Export.Word.Page.WordRenderingExtension();
GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider outputProvider = new
GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider(outputDirectory,
System.IO.Path.GetFileNameWithoutExtension(outputDirectory.Name));

// Overwrite output file if it already exists.
outputProvider.OverwriteOutputFile = true;

reportDocument.Render(wordRenderingExtension, outputProvider, wordSetting);
```

Word Rendering Extension Properties

ActiveReports offers several options to control how reports render to Microsoft Word.

Common properties (HTML and OOXML)

Property	Description
Author ('Author Property' in the on-line documentation)	Sets the name of the author that appears in the Author field of the Properties dialog in the rendered Word document.
FileFormat ('FileFormat Property' in the on-line documentation)	Sets the output file format to HTML (.Doc) or OOXML (.Docx). By default the file format is set to HTML format.
Title ('Title Property' in the on-line documentation)	Sets the title for a document that appears in the Title field of properties dialog in the rendered Word document.

HTML format

Property	Description
BaseUrl ('BaseUrl Property' in the on-line documentation)	Sets the base URL for any relative hyperlinks that appear in the Hyperlink base field of the Properties dialog in the rendered Word document.
Generator ('Generator Property' in the on-line documentation)	Sets the identity of the document generator in the rendered Word document.
PageHeight ('PageHeight Property' in the on-line documentation)	Sets the height of the report pages in inches for the rendered Word document. The value in this property overrides the original settings in the report.
PageWidth ('PageWidth Property' in the on-line documentation)	Sets the width of the report pages in inches for the rendered Word document. The value in this property overrides the original settings in the report.
UseMhtOutput	Indicates whether Mht output is to be used for the resultant Word document or not.

('UseMhtOutput Property' in the on-line documentation)	

OOXML format

Property	Description
CompanyName ('CompanyName Property' in the on-line documentation)	Sets the name of the organization or company that appears in the Company field of Properties dialog in the rendered Word document.
DocumentCompatibilityVersion ('DocumentCompatibilityVersion Property' in the on-line documentation)	Sets the compatibility mode of the document to the latest version (Microsoft Word 2013) or to previous versions (Microsoft Word 2007 - 2010) of Word. By default the compatibility version is set to Word2007.
DpiX ('DpiX Property' in the on-line documentation)	Sets the horizontal resolution of the images in the rendered Word document. By default DpiX is set to 96.
DpiY ('DpiY Property' in the on-line documentation)	Sets the vertical resolution of the images in the rendered Word document. By default DpiY is set to 96.
PageOrientation ('PageSettings Property' in the on-line documentation)	Sets a value that specifies whether the document pages should be printed in portrait or landscape in the rendered Word document.
PaperSize ('PageSettings Property' in the on-line documentation)	Sets the paper size for the page.
Password ('SecuritySettings Property' in the on-line documentation)	Sets a password that must be provided to open the rendered Word document.
ReadOnlyRecommended ('SecuritySettings Property' in the on-line documentation)	Sets a value that indicates whether Microsoft Office Word displays a message whenever a user opens the document, suggesting that the document is read-only.
WritePassword ('SecuritySettings Property' in the on-line documentation)	Sets the write password that is required for saving changes in the rendered Word document.
TOCAutoUpdate ('TOCAutoUpdate Property' in the on-line documentation)	Automatically updates the TableOfContents control while opening the Word document. By default TOCAutoUpdate is set to False.

Limitations

HTML format

- Although background colors for controls export to Word documents, background colors for sections such as Body and Page Header or Footer do not.
- The **BackgroundImage** is not supported when used in TextBox and CheckBox controls embedded in data regions such as Table and Tablix.
- The **BackgroundImage** is not supported for reports, and for **List**, **Container**, **Shape**, **FormattedText**, **Table**, and **Tablix** report controls.
- **KeepTogether** property of Table/Tablix is not supported.
- Some FormattedText tags, for example bi and <s>es</s>, are not exported to Word.
- Image alignments other than the defaults (HorizontalAlignment: Left and VerticalAlignment: Top) are not supported.
- Checkbox color does not affect the color of the square.

OOXML format

Report properties

- The **LineSpacing** property of a report's style sheet, **StartPageNumber** property of the report, **PrintOnLastPage** property of the PageHeader and PageFooter are not supported.
- For Page reports, some of the **NumberingStyle** property (**DocumentMap** settings) options are not supported. The supported **NumberingStyle** options are **Decimal**, **DecimalZero**, **LowerLetter**, **UpperLetter**, **LowerRoman**, **UpperRoman**.
- Background image is not supported for report item except Shape.
- The **BackgroundRepeat** property of the BackgroundImage is not supported in Page (Page reports) and Body (RDL reports).
- For RDL reports, **Background** and **Border** properties of Page Header or Page Footer are not supported.
- Microsoft Word calculates the columns width by the document width. A RDL report calculates the columns width based on the body width, therefore the width of columns in an exported RDL report may differ from an original RDL report.
- In Microsoft Word, the maximum supported page size is 22 inches (55.87 cm) wide and 22 inches (55.87 cm) high. If an exported report exceeds the maximum size, some data may be lost during export.
- In Microsoft Word, a table can have a maximum of 63 columns. If an exported report table has more than 63 columns, then the table is split and therefore an exported document may differ from an original report.
- A repeated Table Footer (the **RepeatOnNewPage** property of Table Footer) or multiple repeated headers on a single page are not supported.
- The **OverflowPlaceHolder** control is not supported.
- If the **PrintOnFirstPage** property of PageHeader or PageFooter is set to **False**, then both PageHeader and PageFooter will not be available on the first page of the exported document.
- The report data gets rendered only in first theme if a Page report containing multiple themes is exported to Docx format.

Report controls

- The **Inset**, **Outset**, and **Windowsinset** border styles (in **BorderStyle** property) are not supported.
- The **Map**, **Chart**, **Image**, **Barcode**, **SparkLine**, **Bullet**, and **CustomControl** report controls are exported as an image. If a report control uses the BorderColor, BorderStyle or BorderWidth properties, a report is exported as a table.
- The **BorderWidth** property of report controls is not exported as is and may differ from the original BorderWidth value.
- **PageBreaks** are not fully supported. The report contents exported to the Word's table or cell items do not support the page breaks.

- For **Shape** report control, if the **BorderStyle** property is set to **Double**, it is exported as Solid.
- For **Line** control, if **LineStyle** property is set to Double/Transparent, it is exported as Solid.
- For **BandedList** data region, only the BandedList Header is repeated on each page. The BandedList Footer, GroupHeaders and GroupFooters are not supported.
- **Tablix** data region is exported as a single table without horizontal split.
- For **Image** control, **Border** properties and **Padding** properties are not supported if the **Sizing** property is set to **Clip**.
- For **Container** report control, rounding corners (the **RoundingRadius** property) are not supported.
- **RepeatToFill** property for Table and Tablix is not supported.
- Images embedded in a Table data region are not properly supported.
- Pagination is not supported due to difference in the layouts of ActiveReports and Word.
- Page Number in Section (Page N of M(Section)) is not supported.
- **KeepTogether** property of Table/Tablix is not supported.
- Horizontal aligned images may overlap in iWord.
- The properties set to the 'Body' region of a Subreport are not exported.
- If a report contains overlapped report controls, these controls appear side by side in an exported Word document and not overlapped as in the Designer's preview.

- **FormattedText**

  - FormattedText is exported as it is. It does not support all HTML and CSS features.
  - The <a> tag without a href attribute, <abbr> and <q> tags are exported as simple text.
  - For Border Styles - Inset and Outset, the <ul> tags are not exported.
  - The **BackgroundColor**, **BackgroundImage**, **BorderColor**, **BorderStyle** and **BorderWidth** properties are not supported.
  - Anchors with an href attribute are exported as hyperlinks.
  - Headers like h1, h2, etc. are exported as corresponding Microsoft Word built-in header styles.

TableOfContents

  - The **TextAlign**, **DisplayPageNumber**, **TextIndent**, and **Overline TextDecoration** properties are not supported.
  - The **Source** property of the Document Map settings is not fully supported. Only the **Headings Only** option of the **Source** property is supported.
  - If a report uses more than one TableOfContents controls, the properties of the first TableOfContents are applied to the other TableOfContents controls in the exported document.
  - The **FillCharacter** property is exported as dots.
  - The background of TOC control appears black on opening exported file in LibreOffice.

TextBox/CheckBox

  - If the **Format** property is set to **Numeric** or **Date,** the exported TextBox has a right alignment. Other Format values are exported with the left alignment.
  - The **Transparent color** for text is exported as white.
  - The **Underline** for numbered lists, Right-To-Left (RTL) option of the **Direction, Angle**, **ShrinkToFit** and **Overline TextDecoration** properties are not supported.
  - The action **Jump to report** is not supported.
  - The **tb-rl** (vertical text) option of the **WritingMode** property is not supported. TextBoxes with the **WritingMode** property set to tb-rl are exported as lr-tb.
  - The **NoWrap** option of the **WrapMode** property is exported as WordWrap.
  - The **LineSpacing** property of an exported document will differ from the original report. This is because in Microsoft Word, the line spacing is calculated by the font size value of a report control plus the line spacing

value of a report control.
- For **CheckBox** control, the **CheckAlignment** property is exported as MiddleRight for TopRight, MiddleRight, and BottomRight options. Other CheckAlignment options are exported as MiddleLeft.
- **Paddings** exceeding 31 inches is exported as border spaces.
- **Right-To-Left** text direction does not work in the LibreOffice.
- On exporting to Word 2013, when the background (shading) and padding are applied on a paragraph, the padding is also applied to the background, so a gap between the border and the background appears on the left side of the paragraph.
- **CharWrap** property is not supported.
- The fields in a TextBox control are evaluated as follows:
  - PageNumber and TotalPages expressions are exported as special fields, evaluated by a text editor (Word or other).
  - The fields placed in Header or Footer are automatically evaluated.
  - The fields placed in the Body should be re-evaluated manually by clicking 'Update field' from context menu.

Interactivity

HTML format

Reports rendered in a Word format supports both Bookmarks and Hyperlinks. However, if visibility toggling (like in a drill-down report) is essential to your report, it is recommended to use the HTML rendering extension. If a Document map is essential to your report, it is recommended to use the PDF rendering extension.

OOXML format

- **Hyperlinks -** Hyperlinks on TextBox and Image controls are rendered as hyperlinks in the Microsoft Word.
- **Bookmarks** - Bookmarks in the report are rendered as Microsoft Word bookmarks. Bookmark links are rendered as hyperlinks that link to the bookmark labels within the document.
- **TOC AutoUpdate** - TableofContents control in the report is rendered as Microsoft Word table of contents.

Rendering to CSV

Comma-Separated Values (CSV) is a form of structured data in plain text. The text in a CSV file is saved as series of values separated by comma. You can use the **CsvRenderingExtension ('CsvRenderingExtension Class' in the on-line documentation)** to render your report in this format.

The following steps provide an example of rendering a report in the Csv format.

1. Create a new Visual Studio project.
2. In the **New Project** dialog that appears, select **ActiveReports 14 Page Report Application** and specify a name for the project in the Name field.
3. Click **OK** to create a new **ActiveReports 14 Page Report Application**. By default a Page report is added to the project.
4. Add reference to GrapeCity.ActiveReports.Export.Xml package in the project.
5. On the Form.cs or Form.vb that opens, double-click the title bar to create the Form_Load event.
6. Add the following code inside the Form_Load event.

Visual Basic.NET code. Paste INSIDE the Form Load event.

```
' Provide the page report you want to render.
Dim report As New GrapeCity.ActiveReports.PageReport()
Dim reportDocument As New GrapeCity.ActiveReports.Document.PageDocument(report)

' Create an output directory.
Dim outputDirectory As New System.IO.DirectoryInfo("C:\MyCSV")
outputDirectory.Create()

' Provide settings for your rendering output.
Dim csvSettings As New
GrapeCity.ActiveReports.Export.Text.Page.CsvRenderingExtension.Settings()
csvSettings.ColumnsDelimiter = ","
csvSettings.Encoding = System.Text.Encoding.UTF8
csvSettings.NoHeader = "True"
csvSettings.QuotationSymbol = """"c
csvSettings.RowsDelimiter = vbCr & vbLf

' Set the rendering extension and render the report.
Dim csvRenderingExtension As New
GrapeCity.ActiveReports.Export.Text.Page.CsvRenderingExtension()
Dim outputProvider As New
GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider(outputDirectory,
System.IO.Path.GetFileNameWithoutExtension(outputDirectory.Name))

' Overwrite output file if it already exists.
outputProvider.OverwriteOutputFile = True

reportDocument.Render(csvRenderingExtension, outputProvider, csvSettings)
```

C# code. Paste INSIDE the Form Load event.

```
// Provide the page report you want to render.
GrapeCity.ActiveReports.PageReport report = new GrapeCity.ActiveReports.PageReport();
GrapeCity.ActiveReports.Document.PageDocument reportDocument = new
GrapeCity.ActiveReports.Document.PageDocument(report);

// Create an output directory.
System.IO.DirectoryInfo outputDirectory = new System.IO.DirectoryInfo(@"C:\MyCSV");
outputDirectory.Create();

// Provide settings for your rendering output.
GrapeCity.ActiveReports.Export.Text.Page.CsvRenderingExtension.Settings csvSettings =
new GrapeCity.ActiveReports.Export.Text.Page.CsvRenderingExtension.Settings();
csvSettings.ColumnsDelimiter = ",";
csvSettings.Encoding = Encoding.UTF8;
csvSettings.NoHeader = "True";
csvSettings.QuotationSymbol = '"';
csvSettings.RowsDelimiter = "\r\n";
```

```
// Set the rendering extension and render the report.
GrapeCity.ActiveReports.Export.Text.Page.CsvRenderingExtension csvRenderingExtension =
new GrapeCity.ActiveReports.Export.Text.Page.CsvRenderingExtension();
GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider outputProvider = new
GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider(outputDirectory,
System.IO.Path.GetFileNameWithoutExtension(outputDirectory.Name));

// Overwrite output file if it already exists.
outputProvider.OverwriteOutputFile = true;

reportDocument.Render(csvRenderingExtension, outputProvider, csvSettings);
```

Rendering to JSON

JavaScript Object Notation (JSON) is a text-based data format in which the data is stored in the hierarchical form. You can use the **JsonRenderingExtension ('JsonRenderingExtension Class' in the on-line documentation)** to render your report in this format.

The following steps provide an example of rendering a report in the Json format.

1. Create a new Visual Studio project.
2. In the **New Project** dialog that appears, select **ActiveReports 14 Page Report Application** and specify a name for the project in the Name field.
3. Click **OK** to create a new **ActiveReports 14 Page Report Application**. By default a Page report is added to the project.
4. Add reference to GrapeCity.ActiveReports.Export.Xml package in the project.
5. On the Form.cs or Form.vb that opens, double-click the title bar to create the Form_Load event.
6. Add the following code inside the Form_Load event.

Visual Basic.NET code. Paste INSIDE the Form Load event.

```
' Provide the page report you want to render.
Dim report As New GrapeCity.ActiveReports.PageReport()
Dim reportDocument As New GrapeCity.ActiveReports.Document.PageDocument(report)

' Create an output directory.
Dim outputDirectory As New System.IO.DirectoryInfo("C:\MyJSON")
outputDirectory.Create()

' Provide settings for your rendering output.
Dim jsonSettings As New
GrapeCity.ActiveReports.Export.Text.Page.JsonRenderingExtension.Settings()
jsonSettings.Formatted = True

' Set the rendering extension and render the report.
Dim jsonRenderingExtension As New
GrapeCity.ActiveReports.Export.Text.Page.JsonRenderingExtension()
```

```
Dim outputProvider As New
GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider(outputDirectory,
System.IO.Path.GetFileNameWithoutExtension(outputDirectory.Name))

' Overwrite output file if it already exists.
outputProvider.OverwriteOutputFile = True

reportDocument.Render(jsonRenderingExtension, outputProvider, jsonSettings)
```

C# code. Paste INSIDE the Form Load event.

```
// Provide the page report you want to render.
GrapeCity.ActiveReports.PageReport report = new GrapeCity.ActiveReports.PageReport();
GrapeCity.ActiveReports.Document.PageDocument reportDocument = new
GrapeCity.ActiveReports.Document.PageDocument(report);

// Create an output directory.
System.IO.DirectoryInfo outputDirectory = new System.IO.DirectoryInfo(@"C:\MyJSON");
outputDirectory.Create();

// Provide settings for your rendering output.
GrapeCity.ActiveReports.Export.Text.Page.JsonRenderingExtension.Settings jsonSettings =
new GrapeCity.ActiveReports.Export.Text.Page.JsonRenderingExtension.Settings();
jsonSettings.Formatted = true;

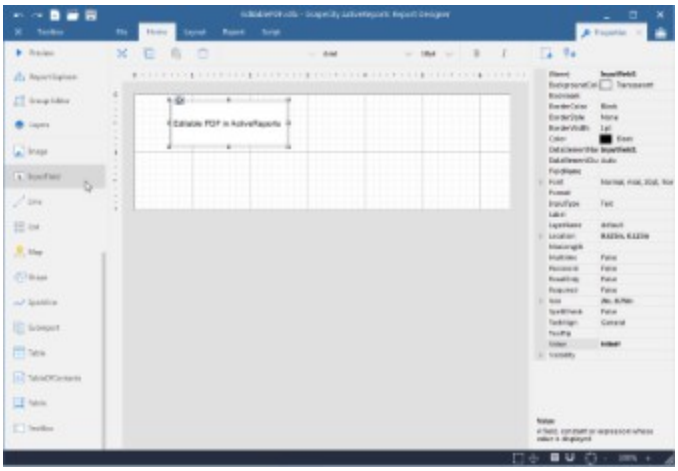
// Set the rendering extension and render the report.
GrapeCity.ActiveReports.Export.Text.Page.JsonRenderingExtension jsonRenderingExtension =
new GrapeCity.ActiveReports.Export.Text.Page.JsonRenderingExtension();
GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider outputProvider = new
GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider(outputDirectory,
System.IO.Path.GetFileNameWithoutExtension(outputDirectory.Name));

// Overwrite output file if it already exists.
outputProvider.OverwriteOutputFile = true;

reportDocument.Render(jsonRenderingExtension, outputProvider, jsonSettings);
```

Editable PDFs

The **InputField** report control provides support for editable fields in an exported PDF report. You can edit text entered into the **Value** property of InputField once a Page or RDL report is exported to PDF.



InputField supports two types – Text and Checkbox, which you can choose in the **InputType** property. Each selected type has its own set of properties.

In case of the Text type, the InputField control gets the set of properties of the TextBox control. If the Checkbox type is selected, then the control inherits the set of properties of the CheckBox control.

Being part of the Professional Edition, the InputField control is not displayed in an exported file with the Standard license.

Export Filters

ActiveReports provides custom components for exporting reports into six formats. Each export format has special features, however, not all formats support all of the features that you can use in your reports. Here are the unique usage possibilities of each format, along with any limitations inherent in each.

- HTML Export: For displaying on Web browsers or e-mail. You can access the HTML Export filter by adding the reference to GrapeCity.ActiveReports.Export.Html.dll in your project.
- PDF Export: For preserving formatting on different computers. You can access the PDF Export filter by adding the reference to GrapeCity.ActiveReports.Export.Pdf.dll in your project.
- Text Export : For transmitting raw data, with little or no formatting. You can access the Text Export filter by adding the reference to GrapeCity.ActiveReports.Export.Xml.dll in your project.
- RTF Export: For preserving some formatting, but allowing reports to be opened with Word or WordPad. You can access the RTF Export filter by adding the reference to GrapeCity.ActiveReports.Export.Word.dll in your project.
- Excel Export: For displaying as spreadsheets. You can access the Excel Export filter by adding the reference to GrapeCity.ActiveReports.Export.Excel.dll in your project.
- TIFF Export: For transmitting fax. You can access the Image Export filter by adding the reference to GrapeCity.ActiveReports.Export.Image.dll in your project.

Some export implementations are also discussed as under:

- Exporting Reports using Export Filters: Elaborates exporting Page Report, RDL Report, or Section Report using the export filters.
- Basic Spreadsheet with SpreadBuilder: Discusses creating Excel spreadsheets cell by cell for maximum control.
- Custom Font Factory (Pro Edition): Explains adding custom fonts in PDFs when using export filters in a Medium trust level environment.
- Font Linking: Explains how to get correct export output when fonts on a deployment machine do not have the glyphs that are used in a development environment.

HTML Export

HTML, or hypertext markup language, is a format that opens in a Web browser. The HTML export filter has a number of useful properties that allow you to control your output. You can set the properties either in code using the **HTMLExport ('HtmlExport Class' in the on-line documentation)** object after adding reference to the following package in your project:

GrapeCity.ActiveReports.Export.Html

HTML Export Properties

Property	Valid Values	Description
BookmarkStyle ('BookmarkStyle Property' in the on-line documentation)	Html (default) or None	Set to Html to generate a page of bookmarks from the bookmarks in the report. If the report has no bookmarks, this setting is ignored.
CharacterSet ('CharacterSet Property' in the on-line documentation)	Big5, EucJp, HzGb2312, Ibm850, Iso2022Jp, Iso2022Kr, Iso8859_1, Iso8859_2, Iso8859_5, Iso8859_6, Koi8r, Ksc5601, ShiftJis, UnicodeUtf16, UnicodeUtf8 (default)	Select the IANA character set that you want to use in the meta tag in the header section of the HTML output. This property only takes effect if the IncludeHtmlHeader property is set to True.
CreateFramesetPage ('CreateFramesetPage Property' in the on-line documentation)	True or False (default)	Set to True to generate a set of frames that display a page of bookmarks (if available) in the left frame and the report document in the right frame. The HTML output uses the specified filename with the extension **.frame.html**.
IncludeHtmlHeader ('IncludeHtmlHeader Property' in the on-line documentation)	True (default) or False	Set to False if you want to embed the HTML output in another HTML document. Otherwise, the HTML output includes the usual HTML, HEAD, and BODY elements.
IncludePageMargins ('IncludePageMargins Property' in the on-line documentation)	True or False (default)	Set to True to include the report's margins in the HTML output.
MultiPage ('MultiPage Property' in the on-line documentation)	True or False (default)	Set to True to create a separate HTML page for each page of the report. Otherwise, the HTML output is a single page.
OutputType ('OutputType Property' in the on-line documentation)	DynamicHtml (default) or LegacyHtml	Set to LegacyHtml to use tables for positioning and avoid the use of cascading style sheets (CSS). Otherwise, positioning of controls is handled in the CSS.
RemoveVerticalSpace ('RemoveVerticalSpace Property' in the on-line documentation)	True or False (default)	Set to True if the OutputType property is set to LegacyHtml and you plan to print the output from a browser. This removes white space from the report to help improve pagination. Otherwise, vertical white

		space is kept intact.
Title ('Title Property' in the on-line documentation)	Any String	Enter the text to use in the header section's title. This is displayed in the title bar of the browser.

More information on output types

By default, the report is exported as DynamicHtml (DHTML), with cascading style sheets (CSS). Using the **OutputType ('OutputType Property' in the on-line documentation)** property, you can change the output to LegacyHtml (HTML). Neither of the output types creates a report that looks exactly like the one you display in the viewer because of differences in formats. Following is the usage of each output type and controls to avoid in each.

DynamicHtml (DHTML)

Usage:

- Create Web reports with Cascading Style Sheets (CSS)
- Open in Web browsers

Does not support:

- Diagonal line control
- Control borders

LegacyHtml (HTML)

Usage:

- Create archival reports
- Open in Web browsers

Does not support:

- Line control
- Control borders
- CrossSectionLine controls
- Overlapping controls
- MinCondenseRate property

Limitations of HTML Export Filter

- Line spacing in exported HTML can be different from the line spacing in Viewer.
- There may be space between the borders of each control in exported file.
- Text in RichTextBox may appear overlapped.
- Vertical Text and Bookmarks are not supported.

PDF Export

PDF, or portable document format, opens in the Adobe Reader. The PDF export filter has a number of useful properties that allow you to control your output. You can set the properties either in code using the **PDFExport ('PdfExport Class'**

in the on-line documentation) object after adding reference to the following package in your project:

GrapeCity.ActiveReports.Export.Pdf

With the PDF export filter, you can use the feature Font Linking.

Note: PDF export and font linking features are only available in the Professional Edition of ActiveReports.

PDF Export Properties

Property	Valid Values	Description
ConvertMetaToPng ('ConvertMetaToPng Property' in the on-line documentation)	True or False (default)	Set to True to change any Windows metafile images to PNG format to keep the file size down. If the report has no metafiles, this setting is ignored.
ExportBookmarks ('ExportBookmarks Property' in the on-line documentation)	True (default) or False	Set to True to generate bookmarks from the bookmarks in the report. If the report has no bookmarks, this setting is ignored. To control how the exported bookmarks are displayed, use Options.DisplayMode detailed below.
FontFallback ('FontFallback Property' in the on-line documentation)	String of font families	Set a comma-delimited string of font families to be used to lookup glyphs missing in the original font.
ImageQuality ('ImageQuality Property' in the on-line documentation)	Lowest, Medium (default), or Highest	Set to Highest in combination with a high value in the ImageResolution property to yield the best printing results when converting Windows metafiles (.wmf). Set to Lowest to keep the file size down. If the report has no metafiles, this setting is ignored.
ImageResolution ('ImageResolution Property' in the on-line documentation)	75 - 2400 dpi	Set to 75 dpi to save space, 150 dpi for normal screen viewing, and 300 dpi or higher for print quality. Use this property in combination with ImageQuality (highest) to yield the best results when the report contains metafiles or the Page.DrawPicture API is used. Neither property has any effect on other image types.
NeverEmbedFonts ('NeverEmbedFonts Property' in the on-line documentation)	A semicolon-delimited string of font names	List all of the fonts that you do not want to embed in the PDF file to keep the file size down. This can make a big difference if you use a lot of fonts in your reports.
Options ('Options Property' in the on-line documentation)	See below	Expand this property to see a group of sub properties. These settings control how the Adobe Reader displays the output PDF file when it is first opened. See the table below for details.
Security ('Security Property' in the on-line documentation)	See below	Expand this property to see a group of sub properties. These settings control encryption and permissions on the output PDF file. See the table below for details.
Signature ('Signature Property' in the on-	A valid PdfSignature object	This must be set up in code. For more information, see Digital Signature Pro .

line documentation)		
Version ('Version Property' in the on-line documentation)	Pdf12, Pdf13, Pdf14, Pdf15, Pdf16, Pdf17, PdfA1a, PdfA1b, PdfA2a, PdfA2b, or PdfA2u	Sets the version of the PDF format the exported document is saved in.

PDF (Portable Document Format)

Usage:

- Create printable reports whose formats do not change from machine to machine.
- Open in Adobe Reader.

Does not support:

- Dash and dot border patterns appear to look longer in the PDF output than in the ActiveReports Window Forms Viewer.
- Multiple lines of vertical text is not supported in Page and RDL reports.
- Transparent background-color in charts are not supported.
- The InputField control is not displayed in an exported file with the Standard license.

Options and Security

When you expand the Options or Security properties in the Properties window, the following sub properties are revealed.

PDF Options Properties

Property	Valid Values	Description
Application ('Application Property' in the on-line documentation)	String	Set to the string value that you want to display in the Adobe Document Properties dialog, Description tab, Application field.
Author ('Author Property' in the on-line documentation)	String	Set to the string value that you want to display in the Adobe Document Properties dialog, Description tab, Author field.
CenterWindow ('CenterWindow Property' in the on-line documentation)	True or False (default)	Set to True to position the Adobe Reader window in the center of the screen when the document is first opened.
DisplayMode	None (default),	Select how to display bookmarks when the document is first opened.

('DisplayMode Property' in the on-line documentation)	Outlines, Thumbs, or FullScreen	• None (default) bookmarks are not displayed until opened by the user. • Outlines shows bookmarks in outline format. • Thumbs shows bookmarks as thumbnails. • FullScreen shows the document in full screen, and bookmarks are not displayed.
DisplayTitle ('DisplayTitle Property' in the on-line documentation)	True or False (default)	Set to True to use the Title string entered in the Title property below. Otherwise, the file name is used.
FitWindow ('FitWindow Property' in the on-line documentation)	True or False (default)	Set to True to expand the window to fit the size of the first displayed page.
HideMenubar ('HideMenubar Property' in the on-line documentation)	True or False (default)	Set to True to hide the menu in the Adobe Reader when the document is first opened.
HideToolbar ('HideToolbar Property' in the on-line documentation)	True or False (default)	Set to True to hide the toolbars in the Adobe Reader when the document is first opened.
HideWindowUi	True or False (default)	Set to True to hide the scrollbars and navigation controls in the Adobe Reader when the document is first opened, displaying only the document.
Keywords ('Keywords Property' in the on-line documentation)	String	Enter keywords to display in the Adobe Document Properties dialog, Description tab, Keywords field.
OnlyForPrint ('OnlyForPrint Property' in the on-line documentation)	True or False (default)	Set to indicate whether the PDF is only for print.
Subject ('Subject Property' in the on-line	String	Enter a subject to display in the Adobe Document Properties dialog, Description tab, Subject field.

documentation)		
Title ('Title Property' in the on-line documentation)	String	Enter a title to display in the Adobe Document Properties dialog, Description tab, Title field. Set DisplayTitle to True to display this text in the title bar of the Adobe Reader when the document is opened.

PDF Security Properties

Property	Valid Values	Description
Encrypt ('Encrypt Property' in the on-line documentation)	True or False (default)	Sets or returns a value indicating whether the document is encrypted.
OwnerPassword ('OwnerPassword Property' in the on-line documentation)	String	Enter the string to use as a password that unlocks the document regardless of specified permissions.
Permissions ('Permissions Property' in the on-line documentation)	None, AllowPrint, AllowModifyContents, AllowCopy, AllowModifyAnnotations, AllowFillIn, AllowAccessibleReaders, or AllowAssembly	Combine multiple values by dropping down the selector and selecting the check boxes of any permissions you want to grant. By default, all of the permissions are granted.
Use128Bit ('Use128Bit Property' in the on-line documentation)	True (default) or False	Set to False to use 40 bit encryption with limited permissions. (Disables AllowFillIn, AllowAccessibleReaders, and AllowAssembly permissions.)
UserPassword ('UserPassword Property' in the on-line documentation)	String	Enter the string to use as a password that unlocks the document using the specified permissions. Leave this value blank to allow anyone to open the document using the specified permissions.

PDF Print Presets Properties

ActiveReports allows you to preset the printing properties for PDF report exports using the **PrintPresets ('PrintPresets Class' in the on-line documentation)** class. This prepopulates the print settings in the Print dialog box. Please see Use PDF Printing Presets for more information.

> **Note:** The print preset properties are only available with the Professional Edition license. An evaluation message is displayed when used with the Standard Edition license.

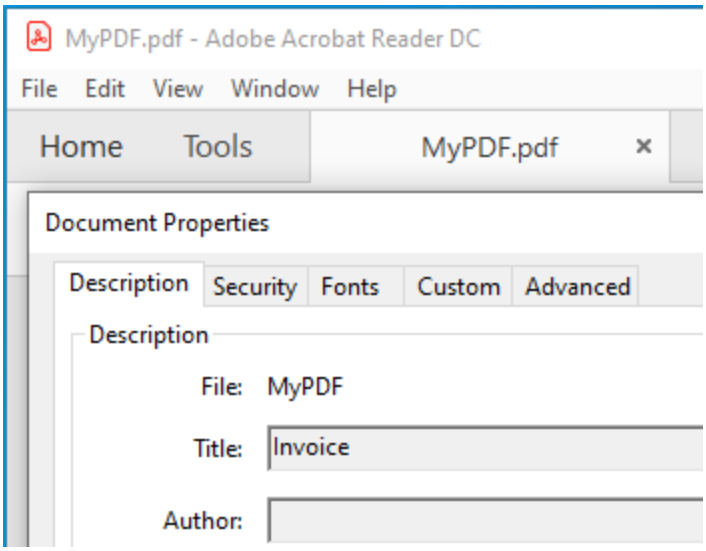
Property	Description
PageScaling ('PageScaling Property'	Specify scaling for the printable area.You can select Default to shrink to the printable area, or you can select None for the actual size.

in the on-line documentation)	
DuplexMode ('DuplexMode Property' in the on-line documentation)	Specify the duplex mode of the printer. For the best results with the duplex option, the selected printer should support duplex printing. You can choose from the following values, • Simplex: Prints on one side of the paper. This is the default value. • Duplex (Flip on long edge): Prints on both sides of the paper with paper flip on the long edge. • Duplex (Flip on short edge): Prints on both sides of the paper with paper flip on the short edge.
PaperSourceByPageSize ('PaperSourceByPageSize Property' in the on-line documentation)	Determines the output tray based on PDF page size, rather than page setting options. This option is useful when printing PDFs with multiple page sizes, where different sized output trays are available. By default, this option is set to False.
PrintPageRange ('PrintPageRange Property' in the on-line documentation)	Specify the range of page numbers as 1-3 or 1, 2, 3.
NumberOfCopies ('NumberOfCopies Property' in the on-line documentation)	Specify the number of copies to print. You can select any number of copies from 2 to 5, or select Default to specify a single copy.

Note: These properties are available in PDF version 1.7 or higher. The **PageScaling** property is supported in PDF version 1.6.

Metadata in PDFs

Adding Metadata



Metadata such as keywords, descriptions are used by the search engines to narrow down the searches. You can add a number of predefined accessors, such as title, contributors, creators, copyright, description, etc.
using **AdditionalMetadata ('AdditionalMetadata Property' in the on-line documentation)** property. The allowed namespaces are:

- Dublin Core Properties
- XMP Core Properties
- PDF Properties

VB code. Paste INSIDE the Form Load event.

```
Dim sectionReport As GrapeCity.ActiveReports.SectionReport = New
GrapeCity.ActiveReports.SectionReport()
Dim xtr As XmlReader = XmlReader.Create(Application.StartupPath & "\SectionReport1.rpx")
sectionReport.LoadLayout(xtr)
sectionReport.Run()
Dim pdfExport As GrapeCity.ActiveReports.Export.Pdf.Section.PdfExport = New
GrapeCity.ActiveReports.Export.Pdf.Section.PdfExport()
Dim metadata1 = New AdditionalMetadataInfo With {
    .[Namespace] = AdditionalMetadataNamespace.PurlOrg, ' Dublin Core Properties
    .Key = "title",
    .Value = "Invoice"
}
pdfExport.Options.AdditionalMetadata.Add(metadata1)
pdfExport.Export(sectionReport.Document, Application.StartupPath & "\MyPDF.pdf")
```

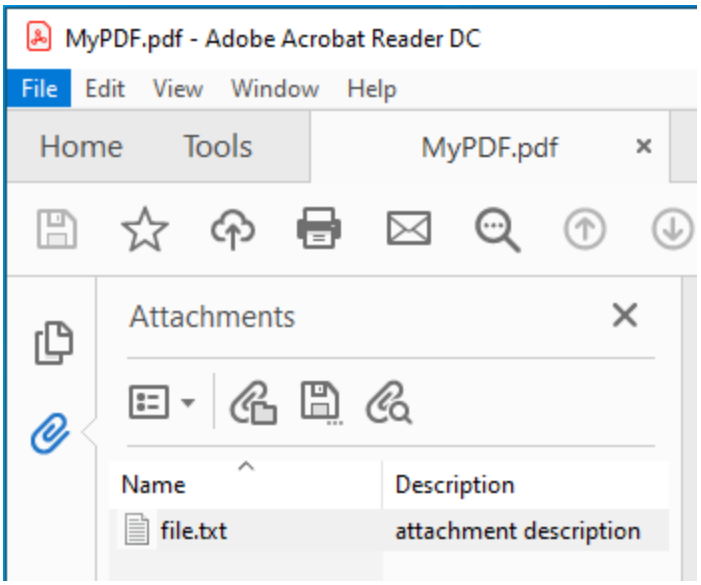
C# code. Paste INSIDE the Form Load event.

```
GrapeCity.ActiveReports.SectionReport sectionReport = new
GrapeCity.ActiveReports.SectionReport();
XmlReader xtr = XmlReader.Create(Application.StartupPath + "\\SectionReport1.rpx");
sectionReport.LoadLayout(xtr);
sectionReport.Run();
GrapeCity.ActiveReports.Export.Pdf.Section.PdfExport pdfExport = new
```

```
GrapeCity.ActiveReports.Export.Pdf.Section.PdfExport();
// Add meta data
var metadata1 = new AdditionalMetadataInfo
{
    Namespace = AdditionalMetadataNamespace.PurlOrg, //Dublin Core Properties
    Key = "title",
    Value = "Invoice"
 };

pdfExport.Options.AdditionalMetadata.Add(metadata1);
pdfExport.Export(sectionReport.Document, Application.StartupPath + "\\MyPDF.pdf");
```

Adding Attachment



You can include an attachment as metadata (such as invoices) to exported PDFs using **Attachments ('Attachments Property' in the on-line documentation)** property. This property allows to attach files such as a .xml or a .txt file in PDF. Below is example to export Section reports to PDF and attach a file to the exported PDF.

VB code. Paste INSIDE the Form Load event.

```
Dim sectionReport As GrapeCity.ActiveReports.SectionReport = New
GrapeCity.ActiveReports.SectionReport()
Dim xtr As XmlReader = XmlReader.Create(Application.StartupPath + _, SectionReport1.rpx)
sectionReport.LoadLayout(xtr)
sectionReport.Run()
Dim pdfExport As GrapeCity.ActiveReports.Export.Pdf.Section.PdfExport = New
GrapeCity.ActiveReports.Export.Pdf.Section.PdfExport()
Dim attachment = New GrapeCity.ActiveReports.Export.Pdf.AttachmentInfo With {
    .Name = "file.txt",
    .Content = System.IO.File.ReadAllBytes("D:\Reports\file.txt"),
```

```
    .Description = "attachment description"
}
pdfExport.Options.Attachments.Add(attachment)
pdfExport.Export(sectionReport.Document, Application.StartupPath & "\MyPDF.pdf")
```

C# code. Paste INSIDE the Form Load event.

```
GrapeCity.ActiveReports.SectionReport sectionReport = new
GrapeCity.ActiveReports.SectionReport();
XmlReader xtr = XmlReader.Create(Application.StartupPath + \\SectionReport1.rpx);
sectionReport.LoadLayout(xtr);
sectionReport.Run();
GrapeCity.ActiveReports.Export.Pdf.Section.PdfExport pdfExport = new
GrapeCity.ActiveReports.Export.Pdf.Section.PdfExport();
// Add attachment
var attachment = new GrapeCity.ActiveReports.Export.Pdf.AttachmentInfo
    {
          Name = "file.txt",
          Content = System.IO.File.ReadAllBytes(@"D:\Reports\file.txt"),
          Description = "attachment description" //optional
    };
pdfExport.Options.Attachments.Add(attachment);
pdfExport.Export(sectionReport.Document, Application.StartupPath + "\\MyPDF.pdf");
```

Open the exported PDF and you should see the attachment. Check the left sidebar in Adobe Acrobat Reader DC.

Note: Metadata in PDFs is part of the Professional Edition. It is supported with the PDF version PDF/A-3b (or higher).

PDF/A Support Limitations

- The **NeverEmbedFonts** property is ignored, so all fonts of a report are embedded into the PDF document.
- The **Security.Encrypt** property is ignored and the PDF export behaves as if this property is always set to **False**.
- The **OnlyForPrint** property is ignored and the PDF export behaves as if this property is always set to **False**.
- **Transparent images** lose their transparency when exported to PDF/A-1.
- **External hyperlinks** are exported as plain text.

Text Export

Plain Text is a format that opens in Notepad or Microsoft Excel depending on the file extension you use in the filePath parameter of the **Export ('Export Method' in the on-line documentation)** method. Use the extension **.txt** to open files in Notepad, or use **.csv** to open comma separated value files in Excel. The Text export filter has a number of useful properties that allow you to control your output. You can set the properties either in code using the **TextExport ('TextExport Class' in the on-line documentation)** object after adding reference to the following package in your project:

- GrapeCity.ActiveReports.Export.Xml

Text Export Properties

Property	Valid Values	Description
Encoding ('Encoding Property' in the on-line documentation)	System.Text.ASCIIEncoding (default), System.Text.UnicodeEncoding, System.Text.UTF7Encoding, or System.Text.UTF8Encoding	This property can only be set in code. Enter an enumerated system encoding value to use for character encoding.
PageDelimiter ('PageDelimiter Property' in the on-line documentation)	String	Enter a character or sequence of characters to mark the end of each page.
QuotationSymbol ('QuotationSymbol Property' in the on-line documentation)	Char	Enter a character to use as quotation mark in the exported text file. Only fields with delimiter characters are quoted.
SuppressEmptyLines ('SuppressEmptyLines Property' in the on-line documentation)	True (default) or False	Set to False if you want to keep empty lines in the exported text file. Otherwise, white space is removed.
TextDelimiter ('TextDelimiter Property' in the on-line documentation)	String	Enter a character or sequence of characters to mark the end of each text field. This is mainly for use with CSV files that you open in Excel.

Text

Usage:

- Create plain text files
- Create comma (or other character) delimited text files
- Feed raw data to spreadsheets or databases
- Open in Notepad or Excel (comma delimited)

Does not support anything but plain fields and labels:

- Supports plain text only with no formatting other than simple delimiters
- Supports encoding for foreign language support

RTF Export

RTF, or RichText format, opens in Microsoft Word, and is native to WordPad. This export does not render reports exactly as they appear in the Viewer due to inherent differences in the formats.

You can set the property either in code using the **RTFExport ('RtfExport Class' in the on-line documentation)** object after adding reference to the following package in your project:

- GrapeCity.ActiveReports.Export.Word

Usage:

- Create word-processing files
- Open in Word or WordPad

Does not support:

- Section or Page back colors
- Angled text

Excel Export

XLSX is a format that opens in Microsoft Excel as a spreadsheet. This export does not render reports exactly as they appear in the Viewer due to inherent differences in the formats. The XLSX export filter has a number of useful properties that allow you to control your output. You can set the properties either in code using the **XLSExport ('XlsExport Class' in the on-line documentation)** object after adding reference to the following package in your project:

- GrapeCity.ActiveReports.Export.Excel

Excel Export Properties

Property	Valid Values	Description
AutoRowHeight ('AutoRowHeight Property' in the on-line documentation)	True or False (default)	Set to True to have Excel set the height of rows based on the contents. Otherwise XlsExport calculates the height of rows. In some cases this may make the output look better inside Excel. However, a value of True may adversely affect pagination when printing, as it may stretch the height of the page.
DisplayGridLines ('DisplayGridLines Property' in the on-line documentation)	True (default) or False	Set to False to hide grid lines in Excel.
FileFormat ('FileFormat Property' in the on-line documentation)	Xls97Plus (default) or Xls95 or Xlsx	Set to Xls95 to use Microsoft Excel 95, Xls95Plus to use Microsoft Excel 97and Xlsx to use Microsoft Excel 2007 or newer.
MinColumnWidth ('MinColumnWidth Property' in the on-line documentation)	Single (VB) or float (C#)	Set the number of inches that is the smallest width for a column in the exported spreadsheet. **Tip**: Larger values reduce the number of empty columns in a sheet. Set this value to 1 inch or more to get rid of small empty columns.
MinRowHeight ('MinRowHeight Property' in the on-line documentation)	Single (VB) or float (C#)	Set the number of inches that is the smallest height for a row in the exported spreadsheet. **Tip**: Larger values force the export to place more controls on a single line by reducing the number of rows added to match blank space. Set this value to .25 inches or more to get rid of small empty rows.
MultiSheet	True or	Set to True to export each page of your report to a separate sheet within the Excel

('MultiSheet Property' in the on-line documentation)	False (default)	file. This can increase performance and output quality at the cost of memory consumption for reports with complex pages and a lot of deviation between page layouts. In general, use False for reports with more than 30 pages.
PageSettings ('PageSettings Property' in the on-line documentation)		Set a print orientation and paper size of Excel sheet.
RemoveVerticalSpace ('RemoveVerticalSpace Property' in the on-line documentation)	True or False (default)	Set to True to remove vertical empty spaces from the spreadsheet. This may improve pagination for printing.
Security ('Security Property' in the on-line documentation)		Set a password and username to protect the excel spreadsheet.
UseCellMerging ('UseCellMerging Property' in the on-line documentation)	True or False (default)	Set to True to merge cells where applicable.
UseDefaultPalette ('UseDefaultPalette Property' in the on-line documentation)	True or False (default)	Set to True to export document with Excel default palette.

Usage:

- Create spreadsheets
- Open in Microsoft Excel

Does not support:

- Line control
- Shapes (other than filled rectangles)
- CrossSectionBox and CrossSectionLine controls
- Overlapping controls
- Borders on controls with angled text
- Angled text
- CheckBox control (only its text element is exported)

TIFF Export

TIFF, or tagged image file format, opens in the Windows Picture and Fax Viewer or any TIFF viewer. This export looks very much like the report as it displays in the viewer, but it is a multi-page image, so the text cannot be edited. The TIFF export filter has a couple of useful properties that allow you to control your output. You can set the properties either in code using the **TIFFExport ('TiffExport Class' in the on-line documentation)** object after adding reference to the following package in your project:

- GrapeCity.ActiveReports.Export.Image

TIFF Export Properties

Property	Valid Values	Description
CompressionScheme ('CompressionScheme Property' in the on-line documentation)	None, Rle, Ccitt3 (default), Ccitt4 or Lzw	Select an enumerated value to use for color output control: • None delivers color output with no compression. • Rle (run-length encoding) is for 1, 4, and 8 bit color depths. • Ccitt3 and Ccitt4 are for 1 color depth, and are used in old standard faxes. • Lzw (based on Unisys patent) is for 1, 4, and 8 bit color depths with lossless compression.
Dither ('Dither Property' in the on-line documentation)	True or False (default)	Set to True to dither the image when you save it to a black and white format (Ccitt3, Ccitt4 or Rle). This property has no effect if the CompressionScheme is set to Lzw or None.
DpiX ('DpiX Property' in the on-line documentation)	Integer (VB) or int (C#) greater than 0	Set the horizontal resolution of a report when exporting to TIFF format. The default value is 200. Setting the DpiX or DpiY property to large values can cause the rendered image to be too large and not enough memory in system can be allocated to the bitmap.
DpiY ('DpiY Property' in the on-line documentation)	Integer (VB) or int (C#) greater than 0	Set the vertical resolution of a report when exporting to TIFF format. The default value is 196. Setting the DpiX or DpiY property to large values can cause the rendered image to be too large and not enough memory in system can be allocated to the bitmap.

Usage:

- Create optical archive reports
- Send reports via fax machines
- Open in image viewers
- Generates an image of each page. 100% WYSIWYG.

Exporting Reports using Export Filters

ActiveReports provides various export filters that can be used to export Page, Section and Rdl Reports to the supported file formats.

Note: In ActiveReports, by default, the NuGet packages are located in the ...\GrapeCity\ActiveReports 14\NuGet folder.

Use the following steps to export reports through export filters. These steps assume that you have already created a Windows Application and added the export controls to your Visual Studio toolbox. For more information, see Adding ActiveReports Controls.

To export a report

1. Create a new project or open an existing Windows Forms Application in Visual Studio.
2. If you are creating a new project, select any of the ActiveReports 14 Page Report, ActiveReports 14 RDL Report, or ActiveReports 14 Section Report (xml-based), in the **New Project** dialog and click OK.

   If you are using an existing project, in the **Solution Explorer**, right-click the project and select **Add** > **New Item.**
3. Select any of the report items from ActiveReports 14 Page Report, ActiveReports 14 RDL Report, or ActiveReports 14 Section Report (xml-based), and click **Add**.
4. Install Export packages from **nuget** as follows:
   i) Go to **Tools** > **Nuget Package Manager** > **Manage Nuget Packages for Solution**...
   ii) Browse the following packages one by one and click Install.
   GrapeCity.ActiveReports.Export.Excel
   GrapeCity.ActiveReports.Export.Html
   GrapeCity.ActiveReports.Export.Image
   GrapeCity.ActiveReports.Export.Pdf
   GrapeCity.ActiveReports.Export.Word
   GrapeCity.ActiveReports.Export.Xml
5. In your project's **Bin>Debug** folder, place the **report.rpx** (Section Report) or **report.rdlx** (Page/Rdl Report).
6. On the Form.cs or Form.vb, double-click the title bar to create the Form_Load event.
7. In **Form_Load event**, add the following code to add a report to your project.

To add Page/Rdl report to your project

Visual Basic.NET code. Paste INSIDE the Form_Load event.

```
' Create a page/Rdl report.
Dim rpt As New GrapeCity.ActiveReports.PageReport()
' Load the report you want to export.
' For the code to work, this report must be stored in the bin\debug folder of your project.
rpt.Load(New System.IO.FileInfo ("report.rdlx"))
Dim MyDocument As New GrapeCity.ActiveReports.Document.PageDocument (rpt)
```

C# code. Paste INSIDE the Form_Load event.

```
// Create a page/Rdl report.
GrapeCity.ActiveReports.PageReport rpt = new GrapeCity.ActiveReports.PageReport();
// Load the report you want to export.
// For the code to work, this report must be stored in the bin\debug folder of your project.
rpt.Load(new System.IO.FileInfo ("report.rdlx"));
GrapeCity.ActiveReports.Document.PageDocument MyDocument = new GrapeCity.ActiveReports.Document.PageDocument
(rpt);
```

To add Section report to your project

Visual Basic.NETcode. Paste INSIDE the Form_Load event.

```
' Create a Section report.
Dim rpt As New GrapeCity.ActiveReports.SectionReport()
' For the code to work, report.rpx must be placed in the bin\debug folder of your project.
Dim xtr As New System.Xml.XmlTextReader(Application.StartupPath + "\report.rpx")
rpt.LoadLayout(xtr)
rpt.Run()
Dim MyDocument As New GrapeCity.ActiveReports.Document.SectionDocument ("rpt")
```

C# code. Paste INSIDE the Form_Load event.

```
// Create a Section report.
GrapeCity.ActiveReports.SectionReport rpt = new GrapeCity.ActiveReports.SectionReport();
// For the code to work, report.rpx must be placed in the bin\debug folder of your project.
System.Xml.XmlTextReader xtr = new System.Xml.XmlTextReader(Application.StartupPath + "\\report.rpx");
```

```
rpt.LoadLayout(xtr);
rpt.Run();
GrapeCity.ActiveReports.Document.SectionDocument MyDocument = new
GrapeCity.ActiveReports.Document.SectionDocument("rpt");
```

8. Add the following code to export Page, Rdl and Section Reports to multiple format.
   This code is common for all the report types (**.rpx** and **.rdlx**).

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the Form_Load event.

```
' Export the report in HTML format.
Dim HtmlExport1 As New GrapeCity.ActiveReports.Export.Html.Section.HtmlExport()
HtmlExport1.Export(MyDocument, Application.StartupPath + "\HTMLExpt.html")

' Export the report in PDF format.
Dim PdfExport1 As New GrapeCity.ActiveReports.Export.Pdf.Section.PdfExport()
PdfExport1.Export(MyDocument, Application.StartupPath + "\PDFExpt.pdf")

' Export the report in RTF format.
Dim RtfExport1 As New GrapeCity.ActiveReports.Export.Word.Section.RtfExport()
RtfExport1.Export(MyDocument, Application.StartupPath + "\RTFExpt.rtf")

' Export the report in text format.
Dim TextExport1 As New GrapeCity.ActiveReports.Export.Xml.Section.TextExport()
TextExport1.Export(MyDocument, Application.StartupPath + "\TextExpt.txt")

' Export the report in TIFF format.
Dim TiffExport1 As New GrapeCity.ActiveReports.Export.Image.Tiff.Section.TiffExport()
TiffExport1.Export(MyDocument, Application.StartupPath + "\TIFFExpt.tiff")

' Export the report in Excel format.
Dim XlsExport1 As New GrapeCity.ActiveReports.Export.Excel.Section.XlsExport()
' Set a file format of the exported excel file to Xlsx to support Microsoft Excel 2007 and newer versions.
XlsExport1.FileFormat = GrapeCity.ActiveReports.Export.Excel.Section.FileFormat.Xlsx
XlsExport1.Export(MyDocument, Application.StartupPath + "\XLSExpt.xlsx")
```

To write the code in C#

C# code. Paste INSIDE the Form_Load event.

```
// Export the report in HTML format.
GrapeCity.ActiveReports.Export.Html.Section.HtmlExport HtmlExport1 = new
GrapeCity.ActiveReports.Export.Html.Section.HtmlExport();
HtmlExport1.Export(MyDocument, Application.StartupPath + "\\HTMLExpt.html");

// Export the report in PDF format.
GrapeCity.ActiveReports.Export.Pdf.Section.PdfExport PdfExport1 = new
GrapeCity.ActiveReports.Export.Pdf.Section.PdfExport();
PdfExport1.Export(MyDocument, Application.StartupPath + "\\PDFExpt.pdf");

// Export the report in RTF format.
GrapeCity.ActiveReports.Export.Word.Section.RtfExport RtfExport1 = new
GrapeCity.ActiveReports.Export.Word.Section.RtfExport();
RtfExport1.Export(MyDocument, Application.StartupPath + "\\RTFExpt.rtf");

// Export the report in text format.
GrapeCity.ActiveReports.Export.Xml.Section.TextExport TextExport1 = new
GrapeCity.ActiveReports.Export.Xml.Section.TextExport();
TextExport1.Export(MyDocument, Application.StartupPath + "\\TextExpt.txt");

// Export the report in TIFF format.
GrapeCity.ActiveReports.Export.Image.Tiff.Section.TiffExport TiffExport1 = new
GrapeCity.ActiveReports.Export.Image.Tiff.Section.TiffExport();
TiffExport1.Export(MyDocument, Application.StartupPath + "\\TIFFExpt.tiff");
```

```
// Export the report in XLSX format.
GrapeCity.ActiveReports.Export.Excel.Section.XlsExport XlsExport1 = new
GrapeCity.ActiveReports.Export.Excel.Section.XlsExport();
// Set a file format of the exported excel file to Xlsx to support Microsoft Excel 2007 and newer versions.
XlsExport1.FileFormat = GrapeCity.ActiveReports.Export.Excel.Section.FileFormat.Xlsx;
XlsExport1.Export(MyDocument, Application.StartupPath + "\\XLSExpt.xlsx");
```

> **Note**: When exporting a report to .MHT file by Export filters use `htmlExport.Export(Document doc, Stream outputStream);`
> For more information, see **HTML Export Method ('Export Method' in the on-line documentation)**.

9. Press **F5** to run the application. The exported files are saved in the bin\debug folder.

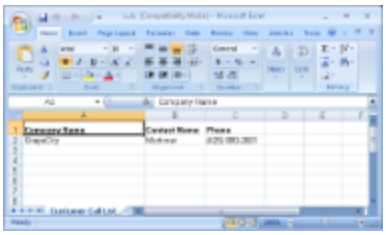
Basic Spreadsheet with SpreadBuilder

Included with the ActiveReports Excel export filter is the SpreadBuilder API. With this utility, you can create Excel spreadsheets cell by cell for maximum control. This walkthrough illustrates how to create a simple custom spreadsheet and save it to an Excel file.

This walkthrough is split into the following activities:

- Adding an ActiveReport to your project
- Adding an GrapeCity.ActiveReports.Export.Excel assembly reference
- Creating a Workbook using code
- Viewing the Excel File

When you have completed this walkthrough, a custom Excel file like the following is created in the **Bin/Debug** subfolder of your project's folder.



To add an GrapeCity.ActiveReports.Export.Excel assembly reference to your project

1. Create a new Visual Studio project.
2. From the Visual Studio **Project** menu, select **Add Reference**.
3. In the Add Reference window that appears, select GrapeCity.ActiveReports.Export.Excel assembly reference and click **OK**.

> **Note**: In ActiveReports, by default, the assemblies are located in NuGet packages are located in the ...\GrapeCity\ActiveReports 14\NuGet folder.

To add code to create a workbook

Double-click the title bar of the Windows Form to create an event-handling method for the Form_Load event. Add code to the handler to:

- Create a Workbook, and add a sheet to the Workbook's Sheets collection
- Set properties on columns and rows in the sheet
- Set values of cells in the sheet

- Use the Save method to create an Excel file

The following example shows what the code for the method looks like.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste inside the form Load event.

```
'Create a Workbook and add a sheet to its Sheets collection
Dim sb As New GrapeCity.SpreadBuilder.Workbook()
sb.Sheets.AddNew()

'Set up properties and values for columns, rows, and cells as desired
With sb.Sheets(0)
   .Name = "Customer Call List" 'sets the name of the sheet
   .Columns(0).Width = 2 * 1440 'sets the width of the 1st column
   .Columns(1).Width = 1440
   .Columns(2).Width = 1440
   .Rows(0).Height = 1440 / 4

'Header row
    .Cell(0, 0).SetValue("Company Name")
    .Cell(0, 0).FontBold = True
    .Cell(0, 1).SetValue("Contact Name")
    .Cell(0, 1).FontBold = True
    .Cell(0, 2).SetValue("Phone")
    .Cell(0, 2).FontBold = True

'First row of data
    .Cell(1, 0).SetValue("GrapeCity")
    .Cell(1, 1).SetValue("Mortimer")
    .Cell(1, 2).SetValue("(425) 880-2601")
End With

'Save the Workbook to an Excel file
sb.Save(Application.StartupPath & "\x.xls")
MessageBox.Show("Your Spreadsheet has been saved to " & Application.StartupPath &
"\x.xls")
```

To write the code in C#

C# code. Paste inside the form Load event.

```
//Create a Workbook and add a sheet to its Sheets collection
GrapeCity.SpreadBuilder.Workbook sb = new GrapeCity.SpreadBuilder.Workbook();
sb.Sheets.AddNew();

//Set up properties and values for columns, rows and cells as desired
sb.Sheets[0].Name = "Customer Call List";
sb.Sheets[0].Columns(0).Width = 2 * 1440;
```

```
sb.Sheets[0].Columns(1).Width = 1440;
sb.Sheets[0].Columns(2).Width = 1440;
sb.Sheets[0].Rows(0).Height = 1440/4;

//Header row
sb.Sheets[0].Cell(0,0).SetValue("Company Name");
sb.Sheets[0].Cell(0,0).FontBold = true;
sb.Sheets[0].Cell(0,1).SetValue("Contact Name");
sb.Sheets[0].Cell(0,1).FontBold = true;
sb.Sheets[0].Cell(0,2).SetValue("Phone");
sb.Sheets[0].Cell(0,2).FontBold = true;

//First row of data
sb.Sheets[0].Cell(1,0).SetValue("GrapeCity");
sb.Sheets[0].Cell(1,1).SetValue("Mortimer");
sb.Sheets[0].Cell(1,2).SetValue("(425) 880-2601");

//Save the Workbook to an Excel file
sb.Save (Application.StartupPath + @"\x.xls");
MessageBox.Show("Your Spreadsheet has been saved to " + Application.StartupPath +
@"\x.xls");
```

To view the Excel File

1. Press **F5** to run the project. A message box informs you of the exact location of the exported x.xls file.
2. Navigate to the Bin/Debug subfolder of your project's folder and open the XLS file.

Custom Font Factory (Pro Edition)

Typically, when using PDF export filters and drawing extensions in a Medium trust level environment or Azure web application, ActiveReports does not have access to the System Fonts folder due to security restrictions. Therefore, if your reports use special glyphs or non-ASCII characters that are only found in certain fonts, the PDF output may be incorrect on some machines.

The ActiveReports custom font factory allows you to embed any fonts you need in PDF in the Medium trust environment. To deploy the necessary fonts with your Medium trust solution, you must add the used font files in the specific folder and set up your web.config file.

ActiveReports looks for the custom fonts in the order as follows:

1. A font specified in the control.
2. A mapped specified font in the **Substitute** setting.
3. A font specified in the the **AddFontLink** setting (Pro Edition only).
4. A font specified in the **SetFallbackFont** setting (Pro Edition only).

Notice that you need to copy the required font files (.ttc, .ttf) manually into the font folder you are accessing.

Custom Font Factory in Windows Azure

For your Azure project, you need to set the properties for all fonts in the project Fonts folder as follows:

1. Set **Copy to Output Directory** to **Copy always.**
2. Set **BuildAction** to **Content**.

To add a font factory section group

XML code. Paste INSIDE the configSections tags of the web.config file.

```
<sectionGroup name="ActiveReports.PdfExport">
<section name="FontFactory"
    type="GrapeCity.ActiveReports.Web.FontFactorySectionHandler,
    GrapeCity.ActiveReports.Web,
    Version=14.0.xxxxx.0, Culture=neutral, PublicKeyToken=cc4967777c49a3ff"
    requirePermission="false" />
</sectionGroup>
```

To create a font factory

Required fonts or the folder containing the required fonts are set in the font factory. Paste code like the following after the configSections tag, but before the appSettings tag.

Note: Please make the required changes for the font that you wish to use. In case of the following sample code, create a Font folder and copy the required font files (arial.ttf, tahoma.ttf, msgothic.ttc, simsun.ttc, gulim.ttc, mingliu.ttc, micross.ttf).

XML code. Paste between configSections and appSettings tags of the web.config file.

```
<ActiveReports.PdfExport>
    <FontFactory Mode="File">
        <AddFolder VirtualPath="~/Fonts" Recurse="true"/>
        <Substitute Font="Helv" To="Helvetica"/>
        <SetFallbackFont Font="Arial"/>

        <!-- font link nodes -->
        <AddFontLink Font="Arial" List="SimSun,gulim,PMingLiU"/>
        <AddFontLink Font="Tahoma" List="MS UI Gothic,SimSun,gulim,PMingLiU"/>
        <AddFontLink Font="MS UI Gothic" List="SimSun,gulim,PMingLiU,Microsoft Sans
Serif" IsDefault="true"/>

    <!-- EUDC link nodes -->
           <DefaultEudcFont File="EUDC.tte"/>
           <AddEudcFont Font="MS UI Gothic" File="myEUDC1.tte"/>
           <AddEudcFont Font="Meiryo" File="myEUDC2.tte"/>
    </FontFactory>
</ActiveReports.PdfExport>
```

Note: For the Azure worker role project, use an absolute path instead of a virtual path in the code above: **<AddFolder Path="~/Fonts" Recurse="true"/>**.

As shown in the previous code, you can embed end-user defined characters (EUDCs) in the PDF output by modifying the web.config file settings by deploying an EUDC file (.tte) to any location.

Configuration settings

EUDC configuration settings

Element	Description
DefaultEUDCFont	This is the node, like the SystemDefaultEudcFont entry in the registry file, that contains the default EUDC font settings.
DefaultEUDCFont File	Specifies the file name of the default EUDC file.
AddEudcFont	This is the node that associates the EUDC file and the fontname. This node can be added more than once.
AddEudcFont Font	Specifies the font name.
AddEudcFont File	Specifies the file name of the EUDC file to associate the above font.

FontFactory

Description: This is the main font factory node to which you can add fonts.

Attributes

Element	Description
Mode	Setting the **Mode** attribute to **File** allows to use a file based factory, or remove the attribute for a Windows GDI factory.

Child Elements

None.

Parent Elements

Element	Description
ActiveReports.PdfExport	The assembly that contains the PdfExport namespace (PDF export, document options, and security classes).

Example

```
<FontFactory Mode="File">
```

AddFolder

Description: Adds all TrueType fonts (.ttc, .ttf) from the specified folder.

Attributes

Element	Description
Path	Specifies the absolute path to the folder.
VirtualPath	Specifies the relative path to the folder.
Recurse	When set to **True**, it can read the subfolder. When set to **False**, it cannot read the subfolder.

Child Elements

None.

Parent Elements

Element	Description
FontFactory	This is the main font factory node to which you can add fonts.

Example

```
<AddFolder VirtualPath="~/Fonts" Recurse="true"/>
```

Substitute

Description: Maps an alternate name of fonts to their official names.

Attributes

Element	Description
Font	Specifies the abbreviated font name (e.g. "Helv").
To	Specifies the official font name (e.g. "Helvetica").

Child Elements

None.

Parent Elements

Element	Description
FontFactory	This is the main font factory node to which you can add fonts.

Example

```
<Substitute Font="Helv" To="Helvetica"/>
```

SetFallbackFont (Professional Edition only)

Description: In the Professional Edition, sets the font to use if a) the specified font is not installed, b) the **Substitute** font is not specified or not installed, or c) the font links are not set or the needed glyphs are not found in the **AddFontLink** setting.

Attributes

Element	Description
Font	Specifies the font name.

Child Elements

None.

Parent Elements

Element	Description
FontFactory	This is the main font factory node to which you can add fonts.

Example

```
<SetFallbackFont Font="Arial"/>
```

AddFontLink (Professional Edition only)

Description: In the Professional Edition, there is the extra support for CJK glyphs. You can add font links that allow the PdfExport to look up any glyphs missing from the specified font in the list of other fonts to check.

Attributes

Element	Description
Font	Specifies the font used in the reports.
List	Specifies the comma-separated list of fonts to be used in case the glyph is not found. ⚠ **Caution:** The character style of the link won't be outputted in case the alternate font file does not exist in the specified folder of the **AddFolder** setting.
IsDefault	When set to **True**, indicates to use the specified list for any fonts that do not have their own

	font links.

Child Elements

None.

Parent Elements

Element	Description
FontFactory	This is the main font factory node to which you can add fonts.

Example

```
<AddFontLink Font="Tahoma" List="MS UI Gothic,SimSun,gulim,PMingLiU"/>
```

Font Linking

Font linking helps resolve the situation when fonts on a deployment machine do not have the glyphs that are used in a development environment. When you find such a glyph mismatch, there is a possibility that the PDF output on development and deployment machines may be different.

In order to resolve this issue, the PDF export filter or the PDF rendering extension looks for the missing glyphs in the installed fonts as follows:

- Checks the system font link settings for each font that is used in the report. HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\FontLink\SystemLink registry key stores the information on font links.
- If font links are not set or the needed glyphs are not found, searches for the glyphs in the fonts declared in the **FontFallback Property (on-line documentation)**.
- Uses glyphs from the font links collection to replace fonts that do not have their own declared linked fonts.
- If necessary glyphs are not found by font links or in the fonts declared in the FontFallback property, glyphs from Microsoft Sans Serif font are taken as the predefined font.

> **Note:** Font Linking is only possible in the Professional Edition of ActiveReports.

Concepts

Learn about concepts that help you to understand how best to use ActiveReports.

Topic	Content
ActiveReports Designer	Learn what each of the tools and UI items on the report designer can help you to accomplish.
ActiveReports Web Designer	Learn about Web Designer in ActiveReports.

Designer Control (Pro Edition)	Learn about Web Designer in ActiveReports.
Standalone Viewers	Learn about the stand-alone designer and Viewer applications that help you create, edit and view a report quickly.
Standalone ActiveReports Designer	Learn about the stand-alone designer and Viewer applications that help you create, edit and view a report quickly.
Page Report/RDL Report Concepts	Learn basic concepts that apply to Page reports and RDL reports.
Section Report Concepts	Learn basic concepts that apply to Section reports.
Visual Query Designer	Learn about Visual Query Designer features and it's SQL capabilities.
Interactive Features	Learn about various interactive features that affect your report output.
Report Parts	Learn about reports and how to use them in different reports.
Common Concepts	Learn how to set different properties to manage appearance of text in different controls.
Localization	Learn about the ActiveReports localization model.
Section 508 Compliance	Learn about Section 508 compliance.

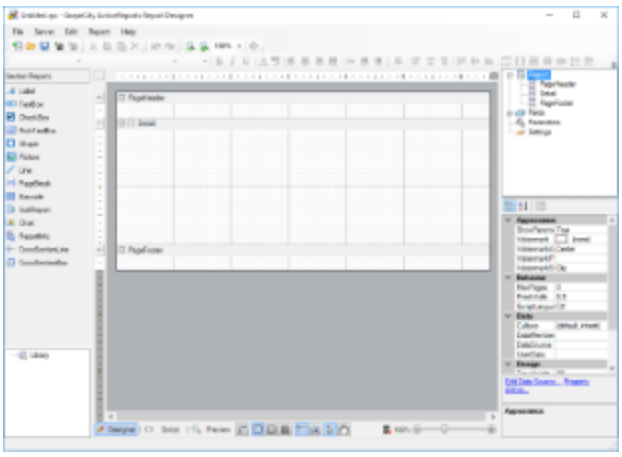
ActiveReports Designer

ActiveReports offers an integrated designer that lets you create report layouts in Visual Studio and edit them at design time, visually, and through code, script, or regular expressions. Like any form in Visual Studio, it includes a Property Window with extensive properties for each element of the report, and also adds its own Toolbox filled with report controls, and a Report Explorer with a tree view of report controls.

The designer supports three types of report layouts: section layout, page layout, and rdl layout.

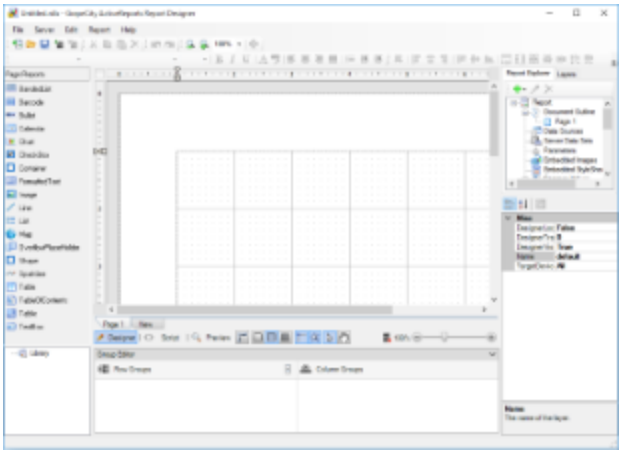
Section Report Layout

This layout presents reports in three banded sections by default: page header, detail and page footer. You can remove the page header and footer, add a report header and footer, and add up to 32 group headers and footers. Drag controls onto these sections to display your report data. Reports designed in this layout are saved in RPX format.



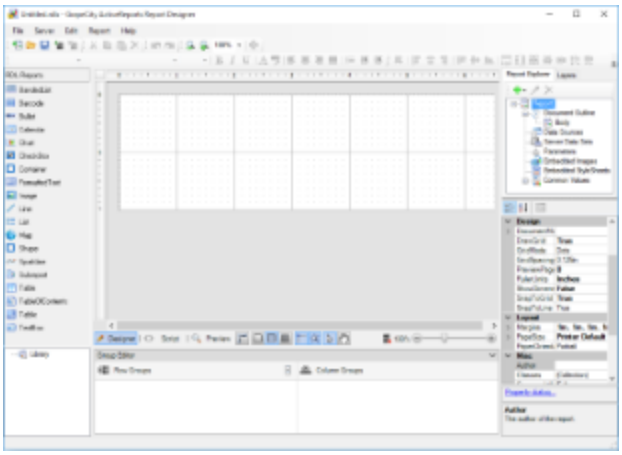
Page Report Layout

This layout defines reports in pages where the same page layout can be used throughout the report or separate layout pages are designed for complex reports. Reports designed in this layout are saved in Rdlx format.



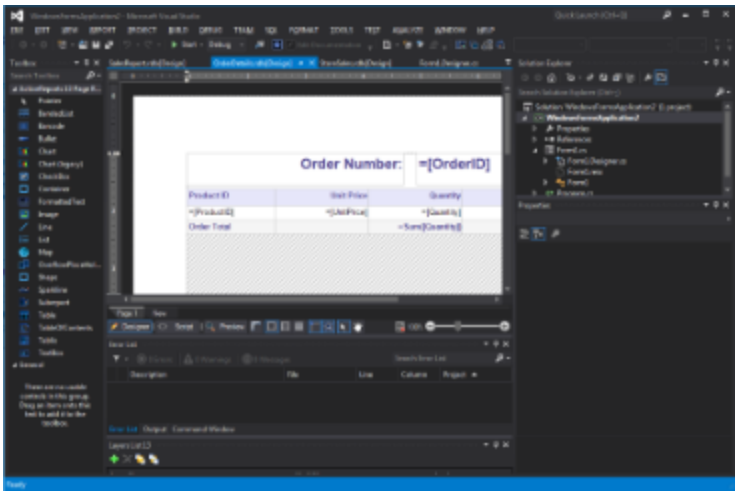
RDL Report Layout

This layout defines reports where controls grow vertically to accommodate data. Reports designed in this layout are saved in Rdlx format.



Note that a theme you select for your Visual Studio, is automatically applied to the ActiveReports Designer. For example, if you select the **Dark** theme for your Visual Studio, this theme is applied automatically to the Designer. The Visual Studio theme is extended to such ActiveReports Designer elements as Reports Library, Layer List, Report Explorer, and Group Editor.

The theme integration works for all reports (Page, RDL, and Section) opened in supported Visual Studio versions.



In this section

Design View
: Explore the elements of the design tab that appear for each report type.

Report Menu
: Learn about the options available in the Report menu in Visual Studio.

Designer Tabs
: Find general information about the Designer, Script, and Preview tabs of the designer.

Designer Buttons
: Learn to control grid settings, drag and drop settings, and mouse modes on the designer.

Page Tabs
: Explore the ways that you can use different page layouts in the same report in Page Reports.

Toolbar
: Learn about the commands available in the ActiveReports Toolbar.

Report Explorer
: Learn how you can use the Report Explorer to manage the report controls, data regions, parameters, and other items in your reports.

Toolbox
: Find information on controls you can use to design report layouts.

Properties Window
: See an overview of how to access properties for report controls, data regions, report sections, and the report itself.

Rulers
: Learn how you can use rulers to align your controls on the report design surface.

Scroll Bars
: See an explanation of scroll bars including the new auto scrolling feature.

Snap Lines
: Find information about snap lines, and how they work.

Zoom Support
: Get the ability to zoom in or zoom out of your report layout.

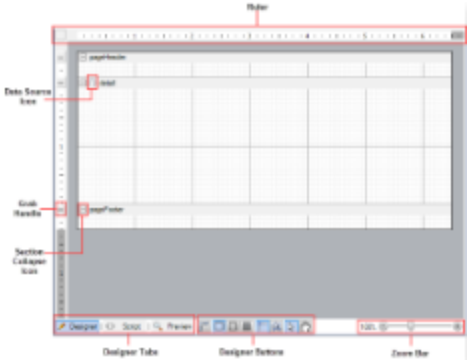
Design View

The report designer is fully integrated with the Microsoft Visual Studio IDE. In this topic, we introduce the main parts

of the designer in Section report, Page report, and RDL report to help you select the one to best suit your specific needs.

Report designer in section reports

In a section report, the designer offers the following features that you can use to create, design and edit a report.



Design Surface

The design surface offers a default report structure that contains a page header, a detail section, and a page footer along with some grey area below these sections. Drag report controls and fields onto these sections to display your data. Use section grab handles to drag a section's height up or down. Right click the report and select **Insert** to add other types of header and footer section pairs.

DataSource Icon

The DataSource icon is located in the band along the top of the detail section. Click this icon to open the **Report Data Source** dialog, where you can bind your report to any OLE DB, SQL, or XML data source. See Report Data Source Dialog for more information

Section Collapse Icon

A Section Collapse icon (-) appears on each band adjacent to the section header. When you click the collapse icon the section collapses and an expand icon (+) appears. Please note that section collapse is only available in the **Designer** tab. All sections of the report are visible in the **Preview** tab or when the report is rendered.

> **Tip:** In order to make a section invisible, set the **Height** property of the section to 0 or the **Visible** property to False.

Rulers

Rulers are located at the top and left of the design view. They help a user visualize the placement of controls in the report layout and how they appear in print. Please note that you have to add the right and left margin widths to determine whether your report fits on the selected paper size. The left ruler includes a grab handle for each section to resize the section height. See Rulers for more information.

Grab Handles

Grab handles on the vertical ruler indicate the height of individual sections. You can drag them up or down to change section heights, or double-click to automatically resize the section to fit the controls in it.

Designer Tabs

The designer provides three tabs: **Designer**, **Script** and **Preview**. You can create your report layout visually in the Designer tab, add script to report events in the Script tab to implement .NET functionality, and see the result in the Preview tab. See Designer Tabs for more information.

Designer Buttons

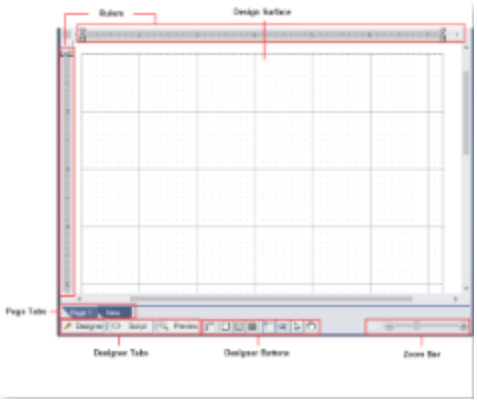
Designer buttons are located below the design surface next to the designer tabs. **Dimension Lines**, **Hide Grid**, **Dots**, **Lines**, **Snap to Lines**, and **Snap to Grid** buttons help you to align report controls and data regions. The **Select Mode** and **Pan Mode** buttons determine whether you select controls on the design surface, or move the visible area of a zoomed-in report. See Designer Buttons for more information.

Zoom Bar

The zoom bar provides a slider that you drag to zoom in and out of the design surface, or you can use the **Zoom in** and **Zoom out** buttons at either end of the slider. See Zoom Support for more information.

Report designer in page reports/RDL reports

In a page report or a RDL report, the designer offers the following features that you can use to create, design and edit a report.



Design Surface

The design surface of a report appears initially as a blank page and grid lines. You can create your own layout and drag report controls and fields onto the design surface to display your data.

Rulers

Use the ruler to determine how your report will look on paper. Please note that you have to add the right and left margin widths to determine whether your report will fit on the selected paper size. See Rulers for more information.

Designer Tabs

The designer provides three tabs: **Designer**, **Script** and **Preview**. You can create your report layout visually in the Designer tab, add script to report events in the Script tab to implement .NET functionality, and see the result in the Preview tab. See Designer Tabs for more information.

Page Tabs(Page Report)

By default, the designer provides two page tabs, **Page 1** and **New**, below the design surface. Each page tab represents a layout page of the report. **Page 1** represents the first page of your report, and you can click **New** to add another page to your report. See Page Tabs for more information.

Designer Buttons

Designer buttons are located below the design surface next to the designer tabs. **Dimension Lines**, **Hide Grid**, **Dots**, **Lines**, **Snap to Lines**, and **Snap to Grid** buttons help you to align report controls and data regions. The **Select Mode** and **Pan Mode** buttons determine whether you select controls on the design surface, or move the visible area of a zoomed-in report. See Designer Buttons for more information.

Zoom Bar

The Zoom Bar provides a slider that you drag to zoom in and out of the design surface, or you can use the **Zoom in** and **Zoom out** buttons at either end of the slider. See Zoom Support for more information.

> **Tip:** ActiveReports provides some useful keyboard shortcuts for the controls placed on the design surface.
>
> - **Arrow Keys:** To move control by one grid line.
> - **[Ctrl] + Arrow Keys:** To move control by 1/100 inch (around 0.025 cms)
> - **[Shift] + Arrow Keys:** To increase or decrease the size of the control by one grid line.

Report Menu

The Report menu provides access to common reporting operations. To show the Report Menu in the Visual Studio menu bar, select the Design View of the report in the ActiveReports Designer. This menu does not appear in the menu bar when the report is not selected.

The following drop-down sections describe the Report menu items. Menu items differ based on the type of report layout in use.

> **Note**: The Report menu in **Visual Studio 2019** is available as submenu under **Extensions**.

Report Menu for Section Reports

Menu Item	Description
Save Layout	Opens the **Save As** dialog to save the newly created report in RPX file format.
Load Layout	Opens the **Open** dialog where you can navigate to any RPX file and open it in the designer. Note that any changes to the current report are lost, as the layout file replaces the current report in the designer.
Data Source	Opens the **Report Data Source** dialog to bind a data source to the report.
Settings	Opens the **Report Settings** dialog. See Report Settings Dialog for more information.
View • **Designer** • **Script** • **Preview**	Opens the Designer, Script or Preview tab. See Designer Tabs for more details.

Report Menu for Page and RDL Reports

Menu Item	Description
Save Layout	Opens the **Save As** dialog to save the newly created report in RDLX file format.
Load Layout	Opens the **Open** dialog where you can navigate to any RDL, RDLX, RDLX-master file and open it in the designer. Note that any changes to the current report are lost, as the layout file replaces the current report in the designer.
Convert to Master	Converts a RDL report to a Master Report.

Report (RDL Reports only)	It disappears from the Report menu when a master report is applied to the report through **Set Master Report**. See Master Reports (RDL) for more details.
Report Parameters	Opens the **Report** dialog to the **Parameters** page where you can manage, add and delete parameters.
Embedded Images	Opens the **Report** dialog to the **Images** page, where you can select images to embed in a report. Once you add images to the collection, they appear in the Report Explorer under the **Embedded Images** node.
Report Properties	Opens the **Report** dialog to the **General** page where you can set report properties such as the author, description, page header and footer properties, and grid spacing.
Stylesheet Editor	Opens the **Stylesheet Editor** dialog, where you can create, edit or remove styles. You can also embed these styles in a style sheet or save them externally in ***.rdlx-styles** format. Embedded style sheets appear under the **Embedded Stylesheets** node in the Report Explorer.
Set Master Report (RDL Reports only) • **Open Local File**	Select Open Local File option to open the **Open** dialog, and then select a master report.
View • **Designer** • **Script** • **Preview**	Opens the Designer, Script or Preview tab. See Designer Tabs for more details.
Page Header (RDL Reports only)	Toggles the report Page Header on or off.
Page Footer (RDL Reports only)	Toggles the report Page Footer on or off.

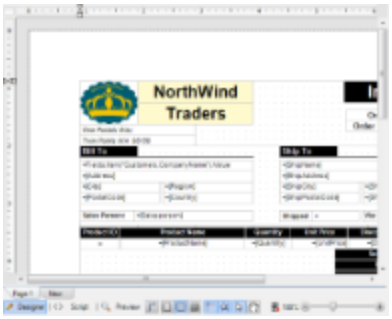
Designer Tabs

The Designer has three tabs located at the bottom of the report design surface. Create a report layout in the Designer tab, write a script in the Script tab to implement .NET functionality and see the result in the Preview tab.

Designer Tab

The Designer tab appears by default on your designer. This tab is used to design your report layout visually. You can implement most of the design-time features here, drag controls from the toolbox to create a layout, bind data regions to data, and set properties for the report and controls through the context menu.

> **Tip:** Layout-related features like designer buttons and zoom slider can be used in this tab to help you manage your report display efficiently.



Script Tab

The Script tab opens the script editor, where you can provide VB.NET or C# functionality to the reports without compiling the .vb or .cs files. You may use either Visual Basic or C# script in this tab with section reports, or Visual Basic with page reports and RDL reports.

The generated reports serve as stand-alone reports which you can load, run, and display in the viewer control without the designer. This feature is useful when distributing reports without recompiling.

In page reports/RDL reports, you can embed code blocks that you can reference in the expressions you use on report controls. See Using Script for more information about using script.

In section reports, you can add code to object events. The two drop-down boxes in the script editor allow you to select any section of the ActiveReport and any events associated with that section, or the report itself and related events. When you select an event, the script editor generates a method stub for the event. See Add Code to Layouts Using Script for more information about scripting in section reports.

Preview Tab

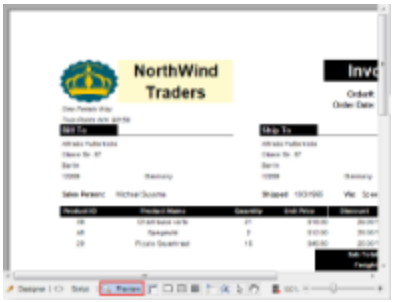
The **Preview tab** allows you to view your report without the need to actually run your project. This makes it easy to quickly see the run-time impact of changes you make in the designer or the code.

This tab does not display data in the following conditions:

- Code or script in the report class is incorrect.
- Report class constructor has been changed.
- Report data source has not been set correctly.
- Settings have been implemented outside the report class
- .mdb file is being copied in the project

When the report is inherited from a class other than ActiveReports, preview is possible only when the base class is in the same project. If the base class is not in the same project and is referencing an external class library, you will not get a preview in the **Preview** tab.

When adding a report directly to ASP.NET Web site, the **Preview** tab is not visible and thus you cannot preview.



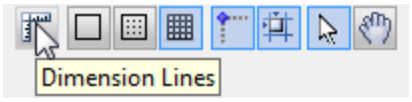
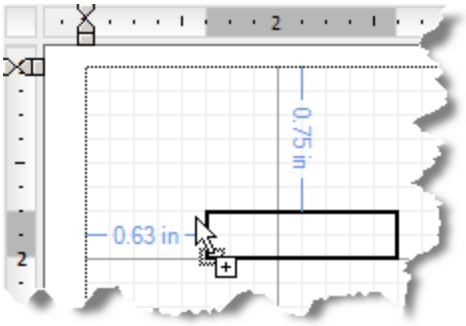
Designer Buttons

Designer buttons are located to the right of the designer tabs along the bottom of the designer, and are enabled when you are on the Designer tab. They allow you to control settings for the design surface.

Grid Settings

Dimension Lines

Dimension lines appear during a drag operation, and run from the borders of the report control or data region being moved or resized to the edges of the report designer surface. Dimension lines let you track the location of the control as you move it by displaying the distance between the control and the edge of the writable area of the report.

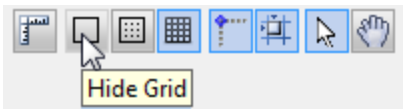
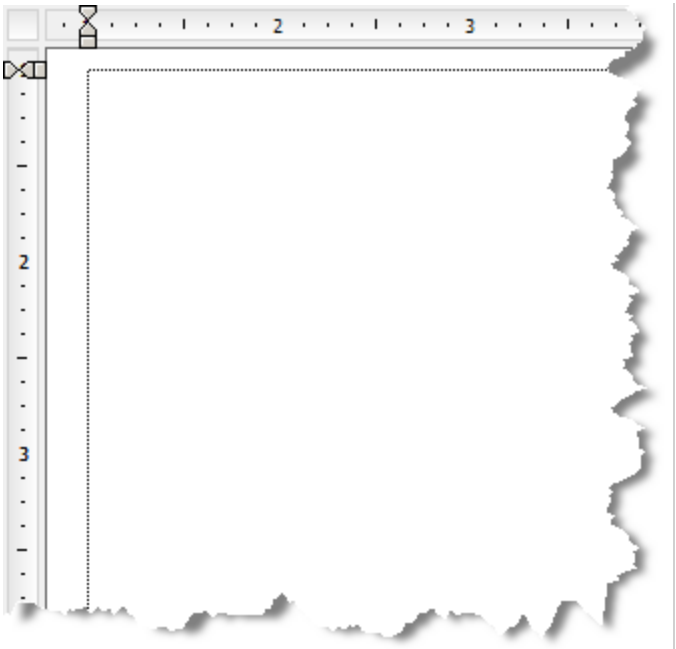
Button Type	Behavior
	

> **Note**: With section reports, you can change the number of grid columns and rows in the Report Settings dialog on the Global Settings tab. With page reports and RDL reports, you can change the grid spacing in the Report Properties dialog on the General tab.

Hide Grid

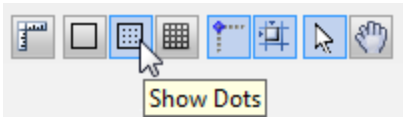
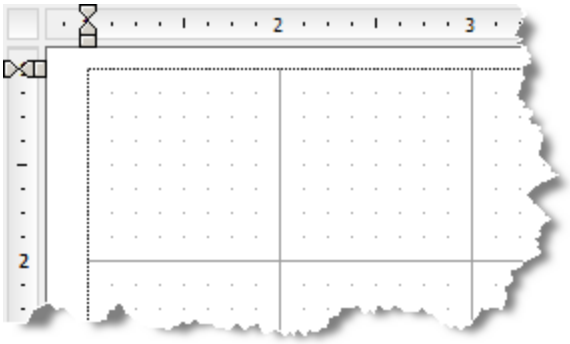
By default, grid lines and dots appear on the report design surface. You can click this button to hide the grid and design your report on a blank page. Lines or dots are also removed from the design surface when you hide the grid, but Snap to Lines or Snap to Grid settings remain unaffected.

Button Type	Behavior

Show Dots

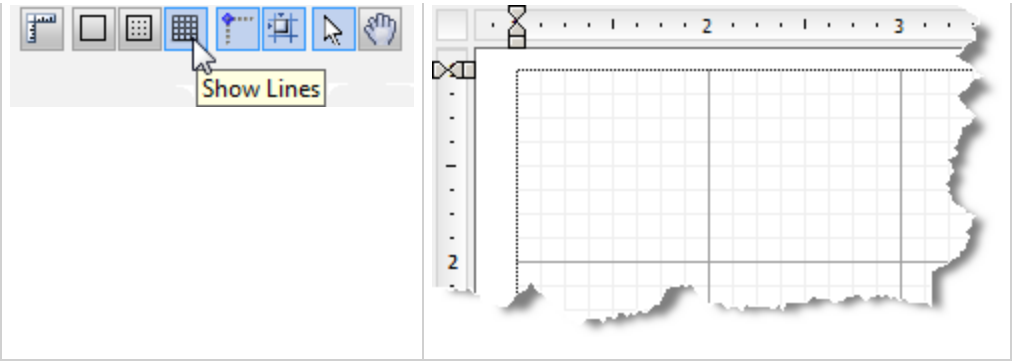
You can click this button to have dots appear on the design surface in between the grid lines to guide you in the placement of controls.

Button Type	Behavior
	

Show Lines

You can click this button to have faint grey lines appear on the design surface in between the grid lines to guide you in the placement of controls.

Button Type	Behavior



Note: Only one option out of Hide Grid, Show Dots and Show Lines can be selected at one time.

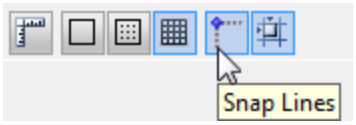
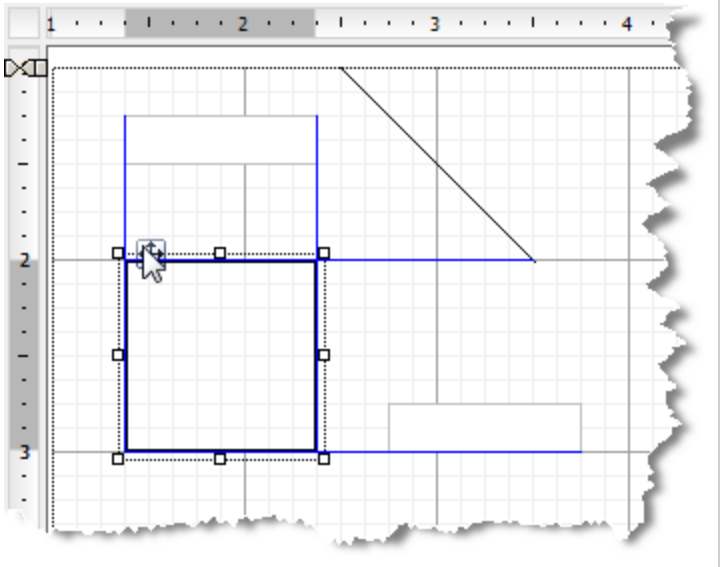
Control Drag and Drop Settings

These settings allow you to specify how you want controls to behave when you drag and drop them on the design surface.

Tip: If you plan to export a report to Excel format, use Snap Lines or Snap to Grid to ensure that your controls are aligned in columns and rows as it prevents overlapping. This makes the export to excel closer to how a report looks at run or design time.

Snap Lines

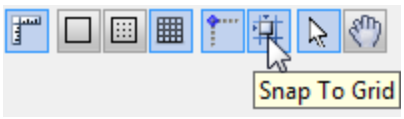
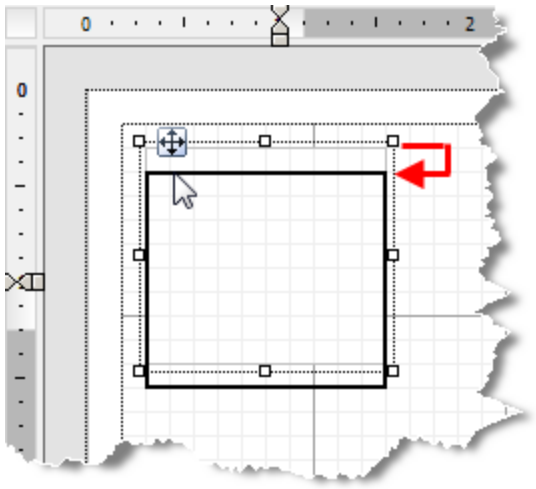
This setting aligns the control you are dragging with other controls on the report design surface. When you drag the control around, snap lines appear when it is aligned with other controls or with the edges of the report, and when you drop it, it snaps into place in perfect alignment. See Snap Lines for more information.

Button Type	Behavior
	

Snap to Grid

This setting aligns the control you are dragging with grid lines on the report design surface. When you drop the control, it

snaps into place in alignment with the nearest grid mark. To place your controls freely on the report design surface, turn this setting off.

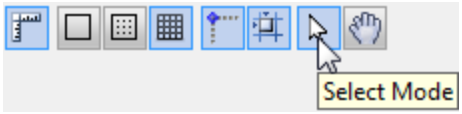
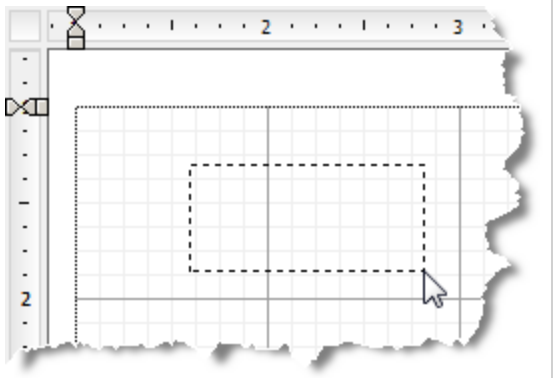
Button Type	Behavior
	

Mouse Modes

These settings allow you to specify how you want the mouse to behave in the designer.

Select Mode

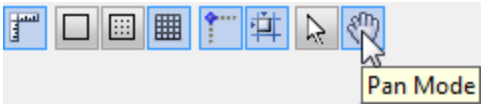
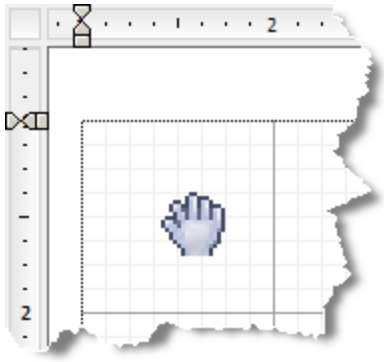
In Select mode, when you click items on the report designer surface, you select them. Use this mode for editing, data binding and styling in the Designer tab. An arrow cursor appears in the Select mode.

Button Type	Behavior
	

Pan Mode

Use the Pan mode to make navigation easier. A hand cursor appears in Pan mode and you can navigate through your report by pressing the left mouse button and dragging the report to the desired position.

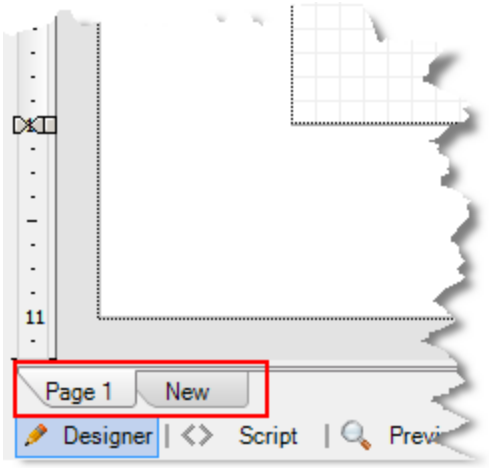
> **Tip:** To enable Pan mode while you are in Select Mode, hold down the middle mouse button and move to the desired location with the hand cursor.

Button Type	Behavior
	

Page Tabs

Page tabs appear in an Excel-like bar below the report design surface. This feature is only available in page reports, where report layouts are designed on separate pages and you can control the way each page appears. Using page tabs, you can select which page to view or edit, add new pages, remove existing pages, reorder pages, and create duplicate pages.

By default, a new report has a **Page 1** tab and a **New** tab.



- **Page 1:** This is the layout for the first page of the report. If no other page layouts exist, the layout on this page is applied to the entire report.
- **New:** Click to add a new page where you can create a layout for pages displayed after the first page.

Right-click any page tab (except the **New** tab) to get a context menu that allows you to **Insert** a new page, **Duplicate** the page, or **Delete** the page.

Adding a new page

To add a new page, click the **New** tab.

A new page tab with an incremented page number appears to the right of any existing page tabs. This page has the same page size and margins as the previous page. The **New** tab moves to the right of the newly added page.

Inserting a page

To insert a page, right-click the page tab and select **Insert**. A page is inserted to the left of the selected page. It has the same page size and margins as the selected page.

Deleting a page

To delete a page, right-click the page tab that you want to remove and select **Delete**. This option is disabled if there is only one page in the report.

Creating a copy of a page

To create a copy of a page, right-click on the page tab that you want to copy and select **Duplicate**. A copy of the selected page appears to the right of the selected page.

> **Note:** When the duplicate page contains a data region, ActiveReports replaces the data region with an OverflowPlaceHolder on the new page. Reset the **GrapeCity.ActiveReports.PageReportModel.DataRegion.OverflowName** property for the duplicated page to maintain the overflow data chain between page tabs.

Reordering pages

To change the order of page tabs, drag a tab and drop it at the desired location. The tab is inserted in the chosen location and the page number is updated according to its position. The page numbers of other tabs also change automatically.

You can cancel the move operation by pressing the **[Esc]** key while dragging the tab.

Toolbar

ActiveReports provides a toolbar integrated with the Visual Studio IDE for quick access to report designing commands. This toolbar is composed of buttons and dropdown lists which offer functions for a number of commonly used commands.

To Show or Hide the Toolbar in Visual Studio

1. Create a new project or open an existing project in Visual Studio.
2. Right click on the Visual Studio toolbar and from the context menu that appears, select **ActiveReports**.

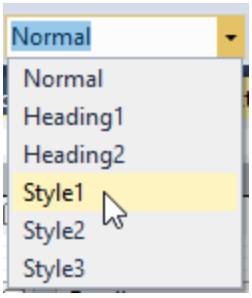
The **ActiveReports** toolbar appears under the Visual Studio menu bar. The toolbar options may differ based on whether you have a section report, page report or a RDL report open.

See the description of each toolbar option in the tables below.

> **Note:** Toolbar descriptions are grouped in a logical order for understanding. The buttons and dropdowns may appear in a different order in the **ActiveReports** toolbar.

Text Decoration

Command	Description
Style	Sets the style for the selected report control.

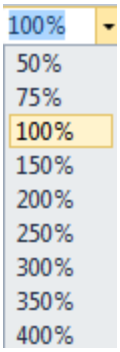
	For details on setting styles in Page and RDL reports, see Working with Styles. For details on setting styles in Section report, see Use External Style Sheets.
Font 	Sets the typeface of all the text in a control. In a section report, for the RichTextBox control, typeface of only the selected text changes. In a page report or a RDL report, in a data region like Tablix or Table, you can change the typeface of the entire data region or only the selected TextBox within the region.
Font Size 	Sets the font size of all the text in a control. In a section report, for the RichTextBox control, font size of only the selected text changes. In a page report or a RDL report, for a data region like Tablix or Table, you can change the font size of the entire data region or only the selected TextBox within the region.
Fore Color 	Opens a **Choose Color** dialog to set the text color of controls.
Back Color 	Opens a **Choose Color** dialog to set the background color of controls.
Bold 	Sets or removes text emphasis from the entire text of the control. In section report, for the RichTextBox control, bold applies to the selected text only. In a page report or a RDL report, for a data region like Tablix or Table, you can change the emphasis of the entire text or only the text of the selected TextBox within the region.
Italic 	Sets or removes text slant for the entire text of the control. In a section report, for the RichTextBox control, italic applies to the selected text only. In a page report or a RDL report, for a data region like Tablix or Table, you can italicize the entire text or only the text of the selected TextBox within the region.
Underline 	Sets or removes the text underline for the entire text of the control. In a section report, for the RichTextBox control, underline applies to the selected text only. In a page report or a RDL report, for a data region like Tablix or Table, you can also

	underline the entire text or only the text of the selected TextBox within the region.

Text Alignment

Command	Description
Align Left 	Aligns the text to the left in the control area.
Center 	Aligns the text to the center in the control area.
Align Right 	Aligns the text to the right in the control area.
Align Justify 	Justifies the text in the control area.

Layout Editing

Command	Description
Zoom Out 	Reduces the magnification level of the design surface and any elements within it.
Zoom In 	Increases the magnification level of the design surface and any elements within it.
Zoom 	Opens a dropdown list to set the magnification level of the design surface between 50% and 400%. Zoom percentage is set to 100% by default.

Control Alignment

Command	Description
Align to Grid 	Snaps the top left of the selected control to the closest gridline.

Align Lefts 	Aligns the selected controls with their left border coinciding with the left border of the primary control. The vertical space separating the controls remains the same.
Align Rights 	Aligns the selected controls with their right border coinciding with the right border of the primary control. The vertical space separating the controls remains the same.
Align Tops 	Aligns the selected controls with their top border coinciding with the top border of the primary control. The horizontal space separating the controls remains the same.
Align Middles 	Aligns the selected controls vertically to the middle with respect to the primary control. The horizontal space separating the controls remains the same.
Align Bottoms 	Aligns the selected controls with their bottom border coinciding with bottom border of the primary control. The horizontal space separating the controls remains the same.

Control Resizing

Command	Description
Make Same Width 	Resizes the width of the selected controls to the width of the primary control.
Make Same Height 	Resizes the height of the selected controls to the height of the primary control.
Make Same Size 	Resizes the size (width and height) of the selected controls to the size of the primary control.
Size to Grid 	Snaps the selected control to the closest gridline by resizing the control on all four sides.

Control Spacing

Command	Description
Make Horizontal Spacing Equal 	Creates equal space between the selected controls with respect to the primary control, using the outermost edges of the controls as end points.
Increase Horizontal Spacing 	Increases the horizontal spacing by one grid unit with respect to the primary control.
Decrease	Decreases the horizontal spacing by one grid unit with respect to the primary control.

Horizontal Spacing 	
Remove Horizontal Spacing 	Removes the horizontal space so that the selected controls move to the nearest edge of the top-left control.
Make Vertical Spacing Equal 	Creates equal space between the selected controls with respect to the primary control, using the top and bottom edges of the control as the end points.
Increase Vertical Spacing 	Increases the vertical spacing by one grid unit with respect to the primary control.
Decrease Vertical Spacing 	Decreases the vertical spacing by one grid unit with respect to the primary control.
Remove Vertical Spacing 	Removes the vertical spacing so that the selected controls move to the nearest edge of the top-left control.

Z-order Alignment

Command	Description
Bring to Front 	Moves the selected controls to the front of all other controls on the report.
Send to Back 	Moves the selected controls behind all other controls on the report.

RichTextBox commands

Command	Description
Bullets 	Adds or removes bullets from the selected text inside a RichTextBox control in a section report.
Increase Indent 	Increases the indent of selected text in the RichTextBox control area in a section report.
Decrease Indent 	Decreases the indent of selected text in the RichTextBox control area in a section report.

Others

Command	Description
View ReportExplorer 	Shows or hides the Report Explorer window. See Report Explorer for further details.
Reorder Groups 	Opens the Group Order dialog, where you can drag and drop groups to rearrange them. This button is enabled when you have multiple groups in a section report.
Layer List 	Opens the Layer List window to displays a list of Layers in the report along with their visibility and lock options. Layer List Explorer also allows you to add or remove Layers from your reports. See Working with Layers for further details.

Note: *Primary control* is the control in a selected group of controls, to which you align all other controls. It is generally the first control selected in the group and has sizing handles (white boxes) which are different from the rest of the selected controls.

Report Explorer

The **Report Explorer** gives you a visual overview of the report elements in the form of a tree view where each node represents a report element.

Using the Report Explorer with any type of report, you can remove controls, add, edit or remove parameters, add a data source, and drag fields onto the report. You can also select the report or any element in the report to display in the Properties Window, where you can modify its properties.

ActiveReports supports three types of reports:

- Section reports (in your choice of XML-based RPX or code-based CS or VB files)
- Page reports (in XML-based RDLX files)
- RDL reports

Section reports, RDL reports and page reports are composed of different types of report elements, so the Report Explorer shows different elements in the report tree depending on the type of report you have open. For more information on how to use the Report Explorer with each, see Exploring Page and RDL Reports and Exploring Section Reports .

To show or hide the Report Explorer in Visual Studio

Once you add the Report Explorer in Visual Studio, it appears every time you create a new Windows application. Use the steps below to hide it when you do not need it.

1. Right-click on the Visual Studio toolbar and select **ActiveReports** to display the report designer toolbar. See Toolbar for further details.
2. On the Designer toolbar, click the **View ReportExplorer** button. The Report Explorer window appears.
3. To hide the Report Explorer, follow the steps above and toggle **View ReportExplorer** back off.

Tip: Another way to show the Report Explorer window in Visual Studio, is from the **View** menu, select **Other Windows**, then **Report Explorer 10**.

To change a report using the Report Explorer

More actions specific to each report type can be found in Exploring Page and RDL Reports and Exploring Section Reports .

To change control properties

1. In the Report Explorer, select the control for which properties are to be changed. In the Properties Window, all of the properties for the item appear.
2. Change property values by entering text or selecting values from drop-down lists. With some properties, for example, **OutputFormat** or **Filters**, when you click the property, an ellipsis button appears to the right. Click the ellipsis button to open a dialog where you can make changes.

To delete a control

1. In the Report Explorer, expand the node that contains the control that you want to remove.
2. Right-click the control and select **Delete**.
3. In the dialog that appears, click **Yes** to confirm the deletion.

Exploring Page and RDL Reports

When you have a Page or RDL report open in the **ActiveReports Designer**, you can see nodes like the following in the Report Explorer.

- Document Outline
  - Each report page (or the body for RDL reports)
    - Each control, for example:
      - BandedList
      - Tablix
      - Table
- Data Sources
  - DataSource (right-click to add a data source; you can have more than one)
    - DataSet (right-click the DataSource to add a data set)
      - Fields (drag onto the report or onto a data region)
    - Another DataSet (you can have more than one)
- Parameters (right-click to open a dialog and add a parameter)
- Embedded Images (right-click to browse for an image to add)
- Embedded StyleSheets (right-click to open and add a style sheet)
- Common Values (drag onto the report to display the value in a textbox)

In the Report Explorer, you can

- remove controls
- add, edit or remove parameters
- add a data source
- drag fields onto the report
- share a data source
- add data sets
- add, edit, or remove embedded images
- drag common values like page numbers, current date, or report name onto the report as a textbox
- select the report or any element in the report to display in the Properties window, where you can modify its

properties.

To add a DataSource

1. In the Report Explorer, right-click the Data Sources node and select **Add Data Source**. The Report Data Source dialog appears, open to the General page.
2. On the General page, drop down the **Type** list and select **Microsoft OleDb Provider**.
3. Under Connection, on the Connection Properties tab, drop down the **OLE DB Provider** list and select **Microsoft.Jet.OLEDB.4.0**.
4. In the **Server or file name** box, enter the path and file name to your Access database, for example, *..\Samples14\Data\NWIND.mdb*.
5. Click the **Accept** button. The new data source is added to the Data Sources node. To use fields from the data source, add a data set.

To share a DataSource

1. In the Report Explorer, expand the DataSources node, right-click the node for the data source that you want to share, and select **Share Data Source**. The Save Shared Data Source File dialog appears.
2. Navigate to the folder where you want to save the file, enter a name for the file, and click **Save**.
3. The type of data source as well as the connection string are saved to a file of type RDSX that you can use in other reports.

To add a DataSet

1. In the Report Explorer, expand the DataSources node, right-click the node for the data source that you want to use, and select **Add DataSet**. The DataSet dialog appears.
2. In the list to the left, select **Query** to show the Query page.
3. In the **Query** box to the right, enter a SQL query to pull the data you want for your report.

   Example Query

   ```
   SELECT * FROM Customers
   ```
4. Click the **Accept** button to create the data set. The data fields appear in the data set node.

To bind a DataSet field to a TextBox control

1. In the Report Explorer, expand the DataSources node, then the node for the data source, then the DataSet that you want to use.
2. From the DataSet node, click the DataSet field that you want to bind to a TextBox control, drag it onto the report surface or onto a data region and drop it.
3. A TextBox control is created and bound to the field with the proper expression in the **Value** property. For example, if you drag the City field onto the report, the Value property of the TextBox contains the expression **=Fields!City.Value**.

To add parameters

1. In the Report Explorer, right-click the **Parameters** node and select **Add Parameter**. The Report Parameters dialog appears.
2. On the General tab of the dialog, enter text for prompting users for a value.
3. On the Available Values tab, you can select values from a DataSet to populate a list from which users can select a value.
4. On the Default Values tab, you can provide default values to use if the user does not select a value.
5. Click **Accept** to save the parameter. The new parameter appears in the Report Explorer under the Parameters node.
6. From the Report Explorer, drag the parameter to report design area to create a TextBox that is bound to the

parameter. When you run the report, the value that the user supplies in the prompt dialog displays in the bound TextBox on the report.

Exploring Section Reports

By default, when you have a section report open in the **ActiveReports Designer**, you can see nodes like the following.

- The Report
  - Each report section, for example:
  - Detail (default, cannot be removed)
  - Report Header and Footer (default, can be removed)
  - Page Header and Footer (can be added)
  - Group Header and Footer (can be added)
    - Each control, for example:
    - TextBox
    - Picture
    - PageBreak
    - SubReport
- Fields
  - Bound (lists fields from the bound data source)
  - Calculated (right-click to add calculated fields)
- Parameters (right-click to add parameters)
- Report Settings (opens a dialog for page setup, printer settings, styles and global settings)

In the **Report Explorer**, in addition to removing controls, adding, editing or removing parameters, adding a data source, and dragging fields onto the report, you can also add, edit, or remove calculated fields; drag bound data fields onto the report as textbox controls; change report settings like margins, printer settings, styles, and ruler and grid settings. You can also select the report or any element in the report to display in the Properties window, where you can modify its properties.

To add a DataSource

1. Click the gray report DataSource icon on the Detail section band to open the Report Data Source dialog.
2. On the OLE DB tab, next to Connection String, click the **Build** button.
3. In the Data Link Properties window that appears, select **Microsoft Jet 4.0 OLE DB Provider** and click the **Next** button. Click the ellipsis (...) button to browse to your database or the sample Northwind database, nwind.mdb.
4. Once you have selected your *.mdb file, click **Open**.
5. Click **OK** to close the window and fill in the Connection String field.
6. In the Query field, enter a SQL query to select the data that you want, for example

   Example Query

   ```
   SELECT * FROM Customers
   ```
7. Click **OK** to save the data source and return to the report design surface. In the Report Explorer, under the Fields node, the Bound node is populated with fields returned by the query.

To add a calculated field

1. In the Report Explorer, expand the **Fields** node.
2. Right-click the **Calculated** node and select **Add**. The new calculated field is displayed in the Report Explorer and in the Properties window.
3. In the Properties window, set the **Formula** property to a calculation, for example: **= UnitPrice * 1.07**

4. Drag the field from the Report Explorer onto the design surface of your report to create a textbox that is bound to the field.

To bind a Field to a TextBox control

1. In the Report Explorer, expand the **Fields** node, then the **Bound** or **Calculated** node that you want to use.
2. Click the field that you want to bind to a TextBox control, drag it onto the report surface and drop it into the section where you want the TextBox to appear.
3. A TextBox control is created and bound to the field with the filed name in the **DataField** property, and a related value in the Name and Text properties. For example, if you drag the City field onto the report, the DataField property of the TextBox becomes **City**, the Name and Text properties become **txtCity1**.

To add parameters

1. In the Report Explorer, right-click the **Parameters** node and select **Add**. The new parameter is displayed in the Report Explorer and in the Properties window.
2. In the Properties window, set the **Prompt** property to a string value to ask users for data.
3. Leave the **PromptUser** property set to **True**. When you run the report, a dialog displays the Prompt to the user.
4. From the Report Explorer, drag the parameter to the report design area to create a TextBox that is bound to the parameter. When you run the report, the value that the user supplies in the prompt dialog displays in the bound TextBox on the report.

To change report settings

1. In the Report Explorer, double-click the **Settings** node. The Report Settings dialog appears.
2. You can set a number of options on the four tabs in the dialog.
3. When you have finished changing report settings, click **OK**.

Toolbox

In ActiveReports, the Visual Studio integrated toolbox tabs display all of the controls specific to the type of report that has focus, or the ActiveReports controls that you can use on Web Forms or Windows Forms.

When a Section report has focus, the **ActiveReports 14 Section Report** toolbox becomes available. For information about the report controls available in this toolbox, please see the Section Report Toolbox topic.

When a Page report has focus, the **ActiveReports 14 Page Report** toolbox becomes available. For information about the report controls available in this toolbox, please see the Toolbox topic.

When a RDL report has focus, the **ActiveReports 14 RDL Report** toolbox becomes available. For information about the report controls available in this toolbox, please see the Toolbox topic.

When a Windows Form has focus, the ActiveReports 14 toolbox group offers the following Windows Forms controls:

- ReportExplorer (requires Professional Edition license)
- Toolbox (requires Professional Edition license)
- Designer (requires Professional Edition license)
- Viewer

When a Web Form has focus, the ActiveReports 14 toolbox group offers one Web control: the WebViewer (requires Professional Edition license). For more information, see Getting Started with the WebViewer .

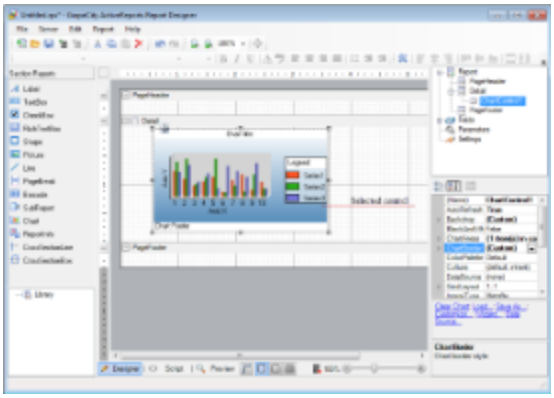
Properties Window

The Visual Studio Properties window is an important tool when you design a report. Select any page, section, data region, control or the report itself to gain access to its properties in the Properties window. By default, this window is placed to the right of the report design area, or wherever you may have placed it in Visual Studio. You can show the list of properties by category or in alphabetical order by clicking the buttons at the top of the Properties window.

Select a property to reveal a description at the bottom of the window. Just above the description is a commands section that contains commands, links to dialogs that give you access to further properties for the item. You can resize the commands or description sections by dragging the top or bottom edges up or down.

> **Tip**: If the commands or description section is missing in Visual Studio, you can toggle it back on by right-clicking anywhere in the Properties window and clicking **Commands** or **Description**.

In the image below, you can see a chart control selected on the designer surface, revealing its properties in the Properties window, along with any associated commands, and a description of the selected property.



Rulers

In ActiveReports, rulers appear to the top and left of the Design View to guide you in vertically and horizontally aligning items in the report. They have large tick marks to indicate half inch points and smaller tick marks to indicate eighths of an inch.

> **Note**: The numbers indicate the distance in inches from the left margin, not from the edge of the page.

Section Reports

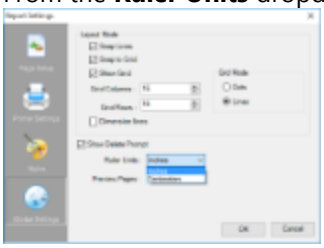
In **Section Reports**, the white area on the ruler indicates the designable area of the report. The grey area at the bottom of the vertical ruler and at the right of the horizontal ruler indicate the report margins. Grab handles on the vertical ruler indicate the height of individual sections. You can drag them up or down to change section heights, or double-click to automatically resize the section to fit the controls in it.

In a section layout, you can change ruler measurements from inches to centimeters and centimeters to inches. Use the following instructions to modify ruler measurements at design- time.

To change ruler measurements at design-time in Section Report

At design time, you can change the ruler measurements from the Report Settings Dialog.

1. In the Report Explorer, double-click the **Settings** node.
2. In the Report Settings dialog that appears, click **Global Settings**.
3. From the **Ruler Units** dropdown select Centimeters or Inches.



In section reports, you can change the units of measure for the rulers. See Change Ruler Measurements for further details.

Page/RDL Reports

In **Page Reports** or **RDL Reports**, margin markers indicate the designable area of the report. The area inside the margin markers is designable, and the areas outside the markers are the margins. To change the margins, you can drag the margin markers to the desired locations.

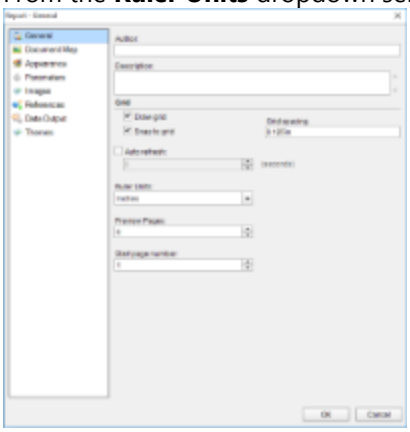
In a page layout, you can change ruler measurements from inches to centimeteres and centimeteres to inches. Use the following instructions to modify ruler measurements in page reports.

To change ruler measurements at design-time in Page Report

At design time, you can change the ruler measurements from the Report Dialog.

1. In the Report Explorer, select the **Report** node.

2. In the command section of the Properties Window, click Property dialog.
3. In the dialog that appears, go to the **General** tab.
4. From the **Ruler Units** dropdown select Centimeters or Inches.



Scroll Bars

Scroll Bars appear automatically when controls or data regions do not fit the visible area of the report design surface. A scroll bar consists of a shaded column with a scroll arrow at each end and a scroll box (also called a thumb) between the arrows. You can scroll up, down, right or left using the scroll arrow buttons, scroll box or mouse wheel.

Auto Scrolling

When a user drags a control beyond the edge of the design surface and the mouse pointer reaches near the surface edge, scrolling starts automatically in the direction of the mouse movement. Auto scrolling works in all four directions. This feature is useful while designing reports with a magnified design view.

> **Note:** In Section Layout and Report Definition Language (RDL), when the mouse button is released during auto scrolling at a location outside the design surface, the surface extends to accommodate the control.

Scrolling stops in the following scenarios:

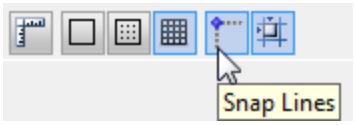
- The user stops dragging the mouse (Mouse Up).
- The user moves the mouse in the opposite direction.
- The **[Esc]** key is pressed while dragging the mouse.

> **Tip:** To enable auto scrolling for multiple controls, hold down the **[Ctrl]** or **[Shift]** key to select the controls. Drag them together to the edge of the design surface and enable auto scrolling.

Snap Lines

Snap lines assist in accurate positioning of elements on a report design surface while you drag report controls on it. These dynamic horizontal and vertical layout guidelines are similar to the ones found in Visual Studio. You can see snap lines on the ActiveReports Designer as well as the Stand-alone designer application.

Snap lines appear on the design surface by default. In order to disable them, click the **Snap Lines** button below the design surface, or in section reports, hold down the **[Alt]** key while dragging a control to temporarily hide the snap lines.



When you drag a control on the design surface, blue snap lines appear and the control slows down as it aligns with another control or a section edge. Unless you are also using the **Snap to Grid** setting, with **Snap Lines**, the control can move freely around the report and can be placed anywhere on the design surface.

> **Tip**: If you plan to export a report to Excel format, use snap lines to ensure that your controls are aligned in columns and rows to avoid empty cells or overlapping of controls in the spreadsheet.

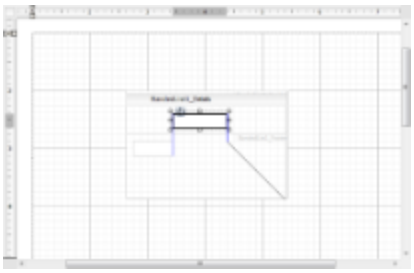
Snap Line Behavior

On dragging with a mouse

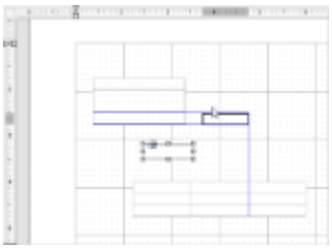
When you drag report controls across the design surface, they snap to other controls, report and section edges. Snap lines appear when the control you are dragging aligns with any edge of any of the following:

- Any control inside any section of the report.

- Another control inside the same data region.

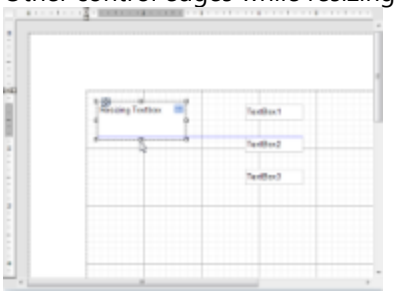


- Parts of a data region (bands in a Banded List, or columns and rows in a Table).

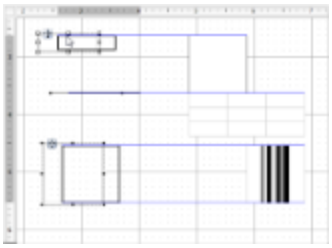


- Report edges and section edges.

- Other control edges while resizing with a mouse.



- On selecting multiple items where all the items move as a single unit, snap lines appear for all items in the selection.



With keyboard actions

- Use [Ctrl] + [Shift] + Arrow keys to resize the selected control from one snap line to the next.
- Use [Ctrl] + Arrow keys to move the selected control to the next snap line.
- Use [Ctrl] + Left mouse button to copy the control and see snap lines appear between the edges of the copied control being dragged and the original control.

> **Note:** Snap lines do not appear when you move a control with arrow keys.

Zoom Support

ActiveReports allows you to zoom in or out on the report design surface for better control over your report layout. As you zoom in or out, the size of every item on the design surface changes.

In the designer, you can access the zoom feature from the Zoom bar below the report design surface where the slider

thumb is set to 100% by default. The slider allows you to zoom in and out of the report designer surface. Using this slider you can magnify the layout from 50% to 400%. You can also use the zoom in (+) and zoom out (-) buttons at either end of the slider to change the zoom level.

Zoom settings are also available on the ActiveReports toolbar where you can change the zoom percentage or use the zoom in/zoom out buttons. See Toolbar for further information.

Keyboard Shortcuts

You can hold down the **Ctrl** key and use the mouse wheel to zoom in and zoom out of the design surface.

You can also use keyboard shortcuts for the following functions:

- **[Ctrl] + [+] :** Zoom in
- **[Ctrl] + [-] :** Zoom out
- **[Ctrl] + 0 :** Return to 100%

ActiveReports Web Designer

The **ActiveReports Web Designer** component can be embedded into web applications to create and edit reports. The following topics explain concepts behind setting up Web Designer and using the API, and finally creating a simple Web Designer sample:

- Set Up Web Designer
- Designer Server API Object
- Designer Options Object
- Create a Simple Web Designer Sample

Set up Web Designer

This document provides information on how to set up the Web Designer.

- **General**
- **Implementation Details**
  - **Supplementary Functions**
  - **On Save Implementation**
  - **On Save As Implementation**
  - **On Open Implementation**
  - **Open File View Implementation**
  - **Open Viewer Implementation**
  - **Open/Close Data Set Picker Implementation**
  - **Open Data Source Editor Implementation**

General

index.html

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>

    <title>ActiveReports Web Designer</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta http-equiv="x-ua-compatible" content="ie=edge">

    <!-- Mandatory -->
    <link rel="stylesheet" href="vendor/css/materialdesignicons.min.css" media="all"
type="text/css" />
    <link rel="stylesheet" href="vendor/css/bootstrap.min.css" />
    <link rel="stylesheet" href="vendor/css/font-awesome.min.css">
    <link rel="stylesheet" href="vendor/css/ionicons.min.css">
    <link rel="stylesheet" href="vendor/css/fonts-googleapis.css" type="text/css">
    <!--------------->

    <!-- Optional: Open-Report/Save-Report-As Dialogs -->
    <link rel="stylesheet" href="file-dialog.css" />

    <!-- Optional: Preview Report -->
    <link rel="stylesheet" href="viewer-container.css" type="text/css">

    <!-- Optional: Add Data Sets using Data Set Picker -->
    <link rel="stylesheet" href="data-set-picker.css" />

    <!-- Optional: File View -->
    <link rel="stylesheet" href="file-view.css" />

    <!-- Optional: Add/Edit Data Sources using Data Source Editor -->
    <link rel="stylesheet" href="ar-datasource-editor.css" />

    <!-- Mandatory -->
    <link rel="stylesheet" href="web-designer.css" />

</head>
<body>

    <!-- Mandatory -->
    <script src="vendor/js/jquery.js"></script>
    <script src="vendor/js/bootstrap.min.js"></script>
    <!--------------->

    <!-- Optional: Open-Report/Save-Report-As Dialogs -->
    <script src="file-dialog.js"></script>

    <!-- Optional - Preview Report -->
    <script src="viewer-container.js"></script>
```

```
    <!-- Optional: Add Data Sets using Data Set Picker -->
    <script src="data-set-picker.js"></script>

    <!-- Optional: File View -->
    <script src="file-view.js"></script>

    <!-- Optional: Add/Edit Data Sources using Data Source Editor -->
    <script src="ar-datasource-editor.js"></script>

    <!-- Mandatory -->
    <script src="baseServerApi.js"></script>
    <script src="web-designer.js"></script>
    <!--------------->

    <!-- Mandatory -->
    <div id="designer-id" style="width: 100%; height: 100%;"></div>

    <!-- Optional: File View -->
    <div id="file-view-id" style="width: 100%; height: 100%; display: none;"></div>

    <!-- Optional - Preview Report -->
    <div id="viewer-container-id" style="width: 100%; height: 100%; display: none;">
</div>

    <!-- Optional: Save-Report-As Dialog -->
    <div id="save-as-dialog-id" style="position: absolute;top: 0;left: 0;width:
100%;height: 100%;display: none;z-index: 9999;"></div>

    <!-- Optional: Open-Report Dialog -->
    <div id="open-dialog-id" style="position: absolute;top: 0;left: 0;width:
100%;height: 100%;display: none;z-index: 9999;"></div>

    <!-- Optional: Add Data Sets using Data Set Picker -->
    <div id="data-set-picker-dialog-id" style="position: absolute;top: 0;left: 0;width:
100%;height: 100%;display: none;z-index: 9999;background-color: #dedede;"></div>

    <!-- Optional: Add/Edit Data Sources using Data Source Editor -->
    <div id="data-source-editor-dialog-id"></div>
    <script>

        var designerId = 'designer-id';
        var viewerContainerId = 'viewer-container-id';
        var fileViewId = 'file-view-id';
        var saveAsDialogId = 'save-as-dialog-id';
        var openDialogId = 'open-dialog-id';
        var dataSetPickerDialogId = 'data-set-picker-dialog-id';
        var dataSourceEditorDialogId = 'data-source-editor-dialog-id';
```

```
        /* Implementation Details - See the section below. */

        /* Mandatory: Create Designer Options */
        var designerOptions =
GrapeCity.ActiveReports.WebDesigner.createDesignerOptions(baseServerApi);

        /* Optional: Save Report */
        designerOptions.saveButton.visible = true;
        designerOptions.onSave = onSaveImpl;

        /* Optional: Save Report As */
        designerOptions.saveAsButton.visible = true;
        designerOptions.onSaveAs = onSaveAsImpl;

        /* Optional: Open Report */
        designerOptions.openButton.visible = true;
        designerOptions.onOpen = onOpenImpl;

        /* Optional: File View */
        designerOptions.openFileView = openFileViewImpl;

        /* Optional: Preview Report */
        designerOptions.openViewer = openViewerImpl;

        /* Optional: Add/Edit/Remove Data Sets */
        designerOptions.dataTab.dataSets.canModify = true;

        /* Optional: Add Data Sets using Data Set Picker */
        designerOptions.dataSetPicker.open = openDataSetPickerImpl;
        designerOptions.dataSetPicker.close = closeDataSetPickerImpl;

        /* Optional: Add/Edit/Remove Data Sources */
        designerOptions.dataTab.dataSources.canModify = true;

        /* Optional: Add/Edit Data Sources using Data Source Editor */
        ARDataSourceEditor.init(dataSourceEditorDialogId);
        designerOptions.openDataSourceEditor = openDataSourceEditorImpl;

        /* Mandatory: Render Designer Application */
        GrapeCity.ActiveReports.WebDesigner.renderApplication(designerId,
designerOptions);

    </script>
</body>

</html>
```

Implementation Details

Supplementary Functions

javascript

```
var dialogs = [
    saveAsDialogId, openDialogId,
    dataSetPickerDialogId, dataSourceEditorDialogId,
];

var showElement = function (id) {
    if (!id) return;
    ($('#' + id)).css('display', 'block');
};

var hideElement = function (id) {
    if (!id) return;
    ($('#' + id)).css('display', 'none');
};

var hideDialogs = function () {
    dialogs.forEach(hideElement);
}

var showDesigner = function () {
    var info = GrapeCity.ActiveReports.WebDesigner.api.getReportInfo();
    var isDirty = GrapeCity.ActiveReports.WebDesigner.api.isReportDirty();
    document.title = info.name + (isDirty ? ' *' : '');
    showElement(designerId);
    GrapeCity.ActiveReports.WebDesigner.focus();
};

var makeOptions = function (baseOptions, additionalOptions) {
    return $.extend({}, baseOptions || {}, additionalOptions || {});
}
```

On Save Implementation

javascript

```
/* Optional: Save Report */
var onSaveImpl = function (options) {
    baseServerApi
        .saveExistingReport({ id: options.reportInfo.id, content:
options.reportInfo.content })
        .then(function (saveResult) {
            options.onSuccess({
                id: saveResult.Id,
```

```
                name: options.reportInfo.name
            });
        });
};
```

On Save As Implementation

javascript

```
/* Optional: Save Report As */
var onSaveAsImpl = function (options) {
    showElement(saveAsDialogId);

    fileDialog.createSaveReportAsDialog(saveAsDialogId, {
        locale: options.locale,

        api: {
            getReportsList: function () {
                return baseServerApi.getReportsList().then(function (reportsList) {
                    return reportsList.map(function (reportInfo) {
                        return { path: reportInfo.Name };
                    });
                });
            },
            saveReport: function (saveOptions) {
                return baseServerApi.saveNewReport({ name: saveOptions.path, content:
options.reportInfo.content }).then(function (saveResult) {
                    return { id: saveResult.Id };
                });
            },
        },
        reportInfo: {
            path: options.reportInfo.name,
        },
        onSuccess: function (saveResult) {
            hideElement(saveAsDialogId);
            options.onSuccess({ id: saveResult.id, name: saveResult.path });
        },
        onClose: function () {
            hideElement(saveAsDialogId);
            showDesigner();
        },
    });
};
```

On Open Implementation

javascript

```
/* Optional: Open Report */
var onOpenImpl = function (options) {
    showElement(openDialogId);
    var loadedReportList = null;

    fileDialog.createOpenReportDialog(openDialogId, {
        locale: options.locale,

        api: {
            getReportsList: function () {
                return baseServerApi.getReportsList().then(function (reportsList) {
                    loadedReportList = reportsList;
                    return reportsList.map(function (reportInfo) {
                        return { path: reportInfo.Name };
                    });
                });
            },
            openReport: function (openOptions) {
                return new $.Deferred(function (def) {
                    var reportInfo = (loadedReportList || []).find(function (r) {
                        return r.Name === openOptions.path;
                    });
                    if (!reportInfo) return def.reject();
                    GrapeCity.ActiveReports.WebDesigner.api.openReport({
                        reportInfo: {
                            id: reportInfo._id,
                            name: reportInfo.Name,
                            permissions: ['all'],
                        },
                        onFinish: function () {
                            def.resolve({ id: reportInfo._id });
                        },
                    });
                });
            },
        },
        onSuccess: function (openResult) {
            hideElement(openDialogId);
        },
        onClose: function () {
            hideElement(openDialogId);
            showDesigner();
        },
    });
};
```

Open File View Implementation

javascript

```
/* Optional: File View */
var openFileViewImpl = function (options) {
    hideElement(designerId);
    showElement(fileViewId);

    var fileViewOptions = makeOptions(options, {
        onClose: function () {
            if (options.onClose) options.onClose();
            hideElement(fileViewId);
            showDesigner();
        },
        serverApi: {
            getReportsList: baseServerApi.getReportsList,
            getReportRevisions: baseServerApi.getReportRevisions,
            getDataSetsList: baseServerApi.getDataSetsList,
            getTemplatesList: baseServerApi.getTemplatesList,
            getTemplateContent: baseServerApi.getTemplateContent,
            getTemplateThumbnail: baseServerApi.getTemplateThumbnail,
        },
        createReport: GrapeCity.ActiveReports.WebDesigner.api.createReport,
        openReport: GrapeCity.ActiveReports.WebDesigner.api.openReport,
        saveReport: GrapeCity.ActiveReports.WebDesigner.api.saveReport,
        ignoreCase: true,
        delimiter: '/',
    });

    fileView.renderFileView(fileViewId, fileViewOptions);
};
```

Open Viewer Implementation

javascript

```
/* Optional: Preview Report */
var openViewerImpl = function (options) {
    hideElement(designerId);
    showElement(viewerContainerId);

    var viewerContainerOptions = makeOptions(options, {
        makeViewerUrl: function () {
            var baseUrl = 'http://localhost:58723/preview/';
            var reportId = encodeURIComponent(options.reportInfo.id);
            var language = options.locale;
            return baseUrl + reportId + '?lng=' + language;
```

```
        },
    });
    viewerContainer.renderViewerContainer(viewerContainerId, viewerContainerOptions,
function () {
        var isTemporaryReport = options.reportInfo.isTemporary;
        if (isTemporaryReport) {
            var reportId = options.reportInfo.id;
            baseServerApi.deleteTemporaryReport({ id: reportId });
        }
        hideElement(viewerContainerId);
        showDesigner();
    });
};
```

Open/Close Data Set Picker Implementation

javascript

```
/* Optional: Add Data Sets using Data Set Picker */
var openDataSetPickerImpl = function (options) {
    var openInPanel = options.mode === 'Panel';
    if (!openInPanel) {
        hideDialogs();
        showElement(dataSetPickerDialogId);
    }
    dataSetPicker.renderDataSetPicker(openInPanel ? options.elementId :
dataSetPickerDialogId, makeOptions(options, {
        serverApi: {
            getDataSetsList: baseServerApi.getDataSetsList,
            getDataSourcesAndDataSets: baseServerApi.getDataSourcesAndDataSets,
        },
        onClose: function () {
            options.onClose();
            if (!openInPanel) hideElement(dataSetPickerDialogId);
            dataSetPicker.dispose();
        }
    }));
};
var closeDataSetPickerImpl = function () {
    dataSetPicker.dispose();
};
```

Open Data Source Editor Implementation

javascript

```
/* Optional: Add/Edit Data Sources using Data Source Editor */
var openDataSourceEditorImpl = function (options) {
```

```
    ARDataSourceEditor.open(makeOptions(options, {
        serverApi: {
            testConnection: baseServerApi.testDataSourceConnection,
        },
    }));
};
```

Designer Server API Object

The module **baseServerApi.js** contains certain functions required for Web Designer operation related to reports, data, themes, images, and report templates. These functions are described as follows. For more information, see **Web Designer API (on-line documentation)**.

> **Note**: Certain properties and functions are marked 'Reserved' since these are not relevant in ActiveReports 14 (for example report revisions) but it is still required to have them for proper working of Web Designer. Web Designer is used not only in ActiveReports 14 but also in older version in ActiveReports Server.

Reports

GET REPORTS LIST

Parameters	Return Value	Response Model
getReportsList()	Promise	Array<ReportsListItem>

REPORTS LIST ITEM OBJECT

```
type ReportsListItem = {
  _id: string, // report id
  Name: string, // report name
  IsCpl: boolean, // specifies whether report is RDL or FPL
  Type: 'PageReport', // RESERVED - for now only 'PageReport' is supported
  $effectivePermissions: 'All', // RESERVED - for now only 'All' is supported
}
```

GET REPORT CONTENT

Parameters	Return Value	Response Model
getReportContent(options: **GetReportContentOptions**)	Promise	**Report JSON Model**

GET REPORT CONTENT OPTIONS OBJECT

```
type GetReportContentOptions {
  id: string, // report id
  version: null, // RESERVED - for now only null is supported
}
```

GET REPORT REVISIONS - RESERVED

Parameters	Return Value	Response Model
getReportRevisions(options: **GetReportRevisionsOptions**)	Promise	Array<ReportRevision>

GET REPORT REVISIONS OPTIONS OBJECT

```
type GetReportRevisionsOptions {
id: string, // report id
}
```

REPORT REVISION OBJECT

```
type ReportRevision = {
_id: string, // report id
Name: string, // report name
IsCpl: boolean, // specifies whether report is RDL or FPL
Type: 'PageReport', // RESERVED - for now only 'PageReport' is supported
$effectivePermissions: 'All', // RESERVED - for now only 'All' is supported
}
```

SAVE NEW REPORT

Parameters	Return Value	Response Model
saveNewReport(options: **SaveNewReportOptions**)	Promise	SaveReportResponse

SAVE NEW REPORT OPTIONS OBJECT

```
type SaveNewReportOptions = {
  name: string, // report name
  content: object, // report JSON model
}
```

SAVE REPORT RESPONSE OBJECT

```
type SaveReportResponse = {
  Id: string, // saved report id
}
```

SAVE EXISTING REPORT

Parameters	Return Value	Response Model
saveExistingReport(options: **SaveExistingReportOptions**)	Promise	SaveReportResponse

SAVE EXISTING REPORT OPTIONS OBJECT

```
type SaveExistingReportOptions = {
  id: string, // report id
```

```
  content: object, // report JSON model
}
```

SAVE TEMPORARY REPORT

Parameters	Return Value	Response Model
saveTemporaryReport(options: **SaveTemporaryReportOptions**)	Promise	SaveReportResponse

SAVE TEMPORARY REPORT OPTIONS OBJECT

```
type SaveTemporaryReportOptions = {
  name: string, // report name
  content: object, // report JSON model
}
```

DELETE TEMPORARY REPORT

This is an optional service function that is used outside of Designer. It can be used for deleting temporary reports when Report Viewer is closed and view is switched back to Designer.

Parameters	Return Value	Response Model
deleteTemporaryReport(options: **DeleteTemporaryReportOptions**)	Promise	DeleteReportResponse

DELETE TEMPORARY REPORT OPTIONS OBJECT

```
type DeleteTemporaryReportOptions = {
  id: string, // report id
}
```

DELETE REPORT RESPONSE OBJECT

```
type DeleteReportResponse = {
  Id: string, // deleted report id
}
```

Data

GET DATA SETS LIST

Parameters	Return Value	Response Model
getDataSetsList()	Promise	Array<DataSetsListItem>

DATA SETS LIST ITEM OBJECT

```
type DataSetsListItem = {
  id: string, // data set id
  name: string, // data set name
```

```
  version: null, // RESERVED - for now only null is supported
}
```

GET DATA SOURCES AND DATA SETS

Parameters	Return Value	Response Model
getDataSourcesAndDataSets(options: **GetDataSourcesAndDataSetsOptions**)	Promise	GetDataSourcesAndDataSetsResponse

WHETHER A DATA SOURCE CAN BE EDITED

Parameters	Return Value	Response Model
canEditDataSource(options: **CanEditDataSourceOptions**)	Promise	GetDataSourcesAndDataSetsResponse

GET CANEDITDATASOURCE OPTIONS OBJECT

```
type CanEditDataSourceOptions = {
    /** data source */
    dataSource: DataSource;
    /** data provider of data source */
    dataProvider: string;
};
```

WHETHER A DATASET CAN BE CREATED FOR A SPECIFIC DATA SOURCE

Parameters	Return Value	Response Model
canAddDataSetForDataSource(options: **CanAddDataSetForDataSourceOptions))**	Promise	GetDataSourcesAndDataSetsResponse

GET CANADDDATASETFORDATASOURCE OPTIONS OBJECT

```
type CanAddDataSetForDataSourceOptions = {
    /** data source */
    dataSource: DataSource;
    /** data provider of data source */
    dataProvider: string;
};
```

WHETHER A DATA SET CAN BE EDITED

Parameters	Return Value	Response Model
canEditDataSet:(options: **CanEditDataSetOptions)**	Promise	GetDataSourcesAndDataSetsResponse

GET EDITDATASET OPTIONS OBJECT

```
type CanEditDataSetOptions = {
    /** data set */
    dataSet: DataSet;
    /** data provider of data set's parent data source */
    dataProvider: string;
};
```

RETURNS A DATA SET INITIALIZED FOR A SPECIFIC DATA SOURCE

Parameters	Return Value	Response Model
initDataSetForDataSource:(options: **InitDataSetForDataSourceOptions**)	Promise	GetDataSourcesAndDataSetsResponse

GET INITDATASETFORDATASOURCE OPTIONS OBJECT

```
type InitDataSetForDataSourceOptions = {
    /** data source */
    dataSource: DataSource;
    /** data provider of data source */
    dataProvider: string;
    /** data set name to be initialized */
    dataSetName: string;
};
```

GET PROPERTIES AVAILABLE FOR A SPECIFIC DATA SET

Parameters	Return Value	Response Model
getDataSetProperties:(options: **GetDataSetPropertiesOptions)**	Promise	GetDataSourcesAndDataSetsResponse

GET GETDATASETPROPERTIES OPTIONS OBJECT

```
type GetDataSetPropertiesOptions = {
    /** data provider of data set's parent data source */
    dataProvider: string;
};
```

GET GETDATASETPROPERTIES RESPONSE OBJECT

```
type DataSetProperty = 'name' | 'commandType' | 'query' | 'fields' | 'calculatedFields' |
'parameters' | 'filters'  | 'caseSensitivity' | 'collation' | 'kanatypeSensitivity' |
'widthSensitivity' | 'accentSensitivity';
```

GET SCHEMA FOR A SPECIFIC DATA SET

Parameters	Return Value	Response Model
getDataSetSchema(options: **GetDataSetSchemaOptions**)	Promise	GetDataSourcesAndDataSetsResponse

GET GETDATASETSCHEMA OPTIONS OBJECT

```
type GetDataSetSchemaOptions = {
    /** data set */
    dataSet: DataSet;
    /** data set's parent data source */
    dataSource: DataSource;
};
```

GET GETDATASETSCHEMA RESPONSE OBJECT

```
type GetDataSetSchemaResponse = {
    /** data set fields */
    fields: Array<Field>;
    /** data set query parameters */
    parameters: Array<QueryParameter>;
};
```

GET DATA SOURCES AND DATA SETS

Parameters	Return Value	Response Model
getDataSourcesAndDataSets(options: **GetDataSourcesAndDataSetsOptions**)	Promise	GetDataSourcesAndDataSetsResponse

GET DATA SOURCES AND DATA SETS OPTIONS OBJECT

```
type GetDataSourcesAndDataSetsOptions = {
  dataSetInfo: {
    id: string, // data set id
    name: string, // data set name
    version: null, // RESERVED - for now only null is supported
  },
  dataSourceInfo: object, // RESERVED
  reportDataSets: Array<DataSet>, // data sets used in report
  reportDataSources: Array<DataSource>, // data sources used in report
}
```

GET DATA SOURCES AND DATA SETS RESPONSE OBJECT

```
type GetDataSourcesAndDataSetsResponse = {
  dataSources: Array<DataSource>,
  dataSets: Array<DataSet>,
}
```

****

GET DATA SET CONTENT

This function is intended to be used in **getDataSourcesAndDataSets()**. The result **dataSetContent** object may store information required to create a data set and its parent data source.

Parameters	Return Value	Response Model
getDataSetContent(options: **GetDataSetContentOptions**)	Promise	dataSetContent: object

GET DATA SET CONTENT OPTIONS OBJECT

```
type GetDataSetContentOptions = {
  id: string, // data set id
}
```

DATA SOURCE AND DATA SET OBJECTS

Basic properties of **DataSource** and **DataSet** types are listed. Object models of **DataSource** and **DataSet** correlate with their structure in **RDLX** report definition.

```
type DataSource = {
  Name: string, // data source name
  ConnectionProperties: {
    ConnectString: string, // connection string
    DataProvider: string, // data provider - see AR Documentation
  },
  // ... more properties - see AR Documentation
}
type DataSet = {
  Name: string, // data set name
  Fields: Array<Field>, // data set fields
  Query: {
    DataSourceName: string, // parent data source name
    CommandType: 'Text' | 'StoredProcedure', // query command type
    CommandText: string, // query command text
    QueryParameters: Array<QueryParameter>,
  }
  // ... more properties - see AR Documentation
}
type Field = {
  Name: string, // field name
  // Either DataField or Value needs to be available.
  DataField?: string, // data field name - valid for bound fields
  Value?: string, // field value - valid for calculated fields
  /* Extra optional properties */
```

```
  // default field aggregate - see AR Documentation for more available aggregates
  Aggregate?: 'Count' | 'Sum' | 'Max' | 'Min' /* etc. */,
  // field data type
  DataType?: 'String' | 'Integer' | 'Float' | 'Number' | 'Boolean' | 'DateTime',
}
type QueryParameter = {
  Name: string, // query parameter name
  Value: string, // query parameter value
}
```

OPEN DATA SOURCE EDITOR IMPLEMENTATION

```
/* Optional: Add/Edit Data Sources using Data Source Editor */
var openDataSourceEditorImpl = function (options) {
        ARDataSourceEditor.open(makeOptions(options, {
                serverApi: {
                        testConnection: baseServerApi.testDataSourceConnection,
                },
        }));
};
```

CAN EDIT DATA SOURCE OPTIONS

```
export type CanEditDataSourceOptions = {
        /** data source */
        dataSource: DataSource;

        /** data provider of data source */
        dataProvider: string;
};

export type CanEditDataSetOptions = {
        /** data set */
        dataSet: DataSet;

        /** data provider of data set's parent data source */
        dataProvider: string;
};
```

INIT DATA SET FOR DATA SOURCE OPTIONS

```
export type InitDataSetForDataSourceOptions = {
/** data source */
dataSource: DataSource;

/** data provider of data source */
dataProvider: string;

/** data set name to be initialized */
dataSetName: string;
```

```
};
```

GET DATA SET PROPERTIES OPTIONS

```
export type GetDataSetPropertiesOptions = {
        /** data provider of data set's parent data source */
        dataProvider: string;
};

export type DataSetProperty = 'name' | 'commandType' | 'query' | 'fields' | 'calculatedFields' |
'parameters' | 'filters'
| 'caseSensitivity' | 'collation' | 'kanatypeSensitivity' | 'widthSensitivity' |
'accentSensitivity';
```

GET DATA SET SCHEMA OPTIONS

```
export type GetDataSetSchemaOptions = {
        /** data set */
        dataSet: DataSet;

        /** data set's parent data source */
        dataSource: DataSource;
};
```

GET DATA SET SCHEMA RESPONSE

```
export type GetDataSetSchemaResponse = {
        /** data set fields */
        fields: Array;

        /** data set query parameters */
        parameters: Array;
};
```

OPEN DATA SOURCE EDITOR BASE OPTIONS

```
export type OpenDataSourceEditorBaseOptions = {
        /** locale passed by Designer */
        locale: 'en' | 'ja' | 'zh';


        /** **Non-null** if an existing data source edited, **null** - if a new one is created.
*/
        dataSource: DataSource | null;

        /** data sources used in report */
        reportDataSources: Array;
```

```
        /** callback on successful data source editing/creation */
        onSuccess: (options: DataSourceEditorOnSuccessOptions) => void;

        /** callback on closing Data Source Editor */
        onClose: () => void;
};
```

DATA SOURCE EDITOR ONSUCCESS OPTIONS

```
export type DataSourceEditorOnSuccessOptions = {
        /** successfully created or edited data source */
        dataSource: DataSource;
};
```

OPEN DATA SET PICKER BASE OPTIONS

```
export type OpenDataSetPickerBaseOptions = {
        /** locale passed by Designer */
        locale: 'en' | 'ja' | 'zh';

        /** If mode is **Panel**, Data Set Picker is rendered within **Data** tab. */
        mode: DataSetPickerMode;

        /** If mode is not **Panel**, Data Set Picker is rendered to an element with this id. */
        elementId?: string;

        /** data sources used in report */
        reportDataSources: Array;

        /** data sets used in report */
        reportDataSets: Array;

        /** New data sources and data sets to be added are passed as parameters to this
function. */
        addDataSourcesAndDataSets: (options: AddDataSourcesAndDataSetsOptions) => void;

        /** callback on closing Data Set Picker */
        onClose: () => void;
};
```

DATA SET PICKER MODE

```
export type DataSetPickerMode = 'Panel' | 'Dialog';
```

ADD DATA SOURCES AND DATA SETS OPTIONS

```
export type AddDataSourcesAndDataSetsOptions = {
        /** data sources to-be-added */
        dataSources: Array;
```

```
        /** data sets to-be-added */
        dataSets: Array;
};
```

CAN ADD DATA SET FOR DATA SOURCE OPTIONS

```
export type CanAddDataSetForDataSourceOptions = {
        /** data source */
        dataSource: DataSource;

        /** data provider of data source */
        dataProvider: string;
};
```

DATA SOURCE EDITOR API

```
                    /**
 * Type of **ARDataSourceEditor** object exported by **ar-datasource-editor.js** module.
 */
export type DataSourceEditorApi = {
        /* Initializes **Data Source Editor** using the specified
**DataSourceEditorInitOptions** object and renders it to ** element with id
**dataSourceEditorElementId**.
         *
         * **Example:**
         * ```javascript
         * ARDataSourceEditor.init('data-source-editor-id', dataSourceEditorInitOptions);
         * ```
         *
         * @param dataSourceEditorElementId string
         * @param options DataSourceEditorInitOptions object
         */
        init: (dataSourceEditorElementId: string, options?: DataSourceEditorInitOptions) =>
void;

        /**
         * Opens Data Source Editor.
         *
         * **Example:**
         * ```javascript
         * ARDataSourceEditor.open(dataSourceEditorOptions);
         * ```
         *
         * @param options DataSourceEditorOptions object
         */
        open: (options: DataSourceEditorOptions) => void;
};

export type CustomProviderDescriptor = {
```

```
        /**
         * key - data provider identifier\
         * This value is used for **DataSource.ConnectionProperties.DataProvider** property.
         */
        key: string;

        /**
         * name - data provider label\
         * This value is used as a friendly data provider label in UI.
         */
        name: string;
};

export type DataSourceEditorInitOptions = {
        /** predefined providers */
        predefinedProviders?: Array;

        /** OLE DB providers */
        oleDbProviders?: Array;

        /** custom providers */
        customProviders?: Array;
};

export type PredefinedProvider = 'SQL' | 'OLEDB' | 'ODBC' | 'JSON' | 'CSV' | 'XML';

export type OleDbProvider = PredefinedOleDbProvider | string;

export type PredefinedOleDbProvider = 'Microsoft.Jet.OLEDB.4.0' | 'SQLOLEDB.1' | 'MSDataShape.1'
| 'MSDASQL.1';

export type DataSourceEditorOptions = OpenDataSourceEditorBaseOptions & {
        /** Specifies custom localization data. */
        localeData?: Array;
```

DATA SOURCE EDITOR SERVER API

```
/** **Data Source Editor** server API */
        serverApi: {
                /**
                 * Tests data source connection.
                 *
                 * @param options TestConnectionOptions object
                 */
                testConnection: (options: TestConnectionOptions) => Promise;
        };
};

export type TestConnectionOptions = {
        /** data source */
```

```
        dataSource: DataSource;
};

export type TestConnectionResponse = {
        /** error code - 0 means success */
        Code: number;

        /** databases list available for test connection settings */
        Databases: Array | null,

        /** error message */
        Error: string | null,
};
```

Themes

GET THEMES LIST

Parameters	Return Value	Response Model
getThemesList()	Promise	Array<ThemesListItem>

THEMES LIST ITEM OBJECT

```
type ThemesListItem = {
  _id: string, // theme id
  Name: string, // theme name
  IsDefault: boolean, // specifies whether theme is default or not, only a single theme can be
default
  Dark1: string, // Dark1 theme color
  Dark2: string, // Dark2 theme color
  Light1: string, // Light1 theme color
  Light2: string, // Light2 theme color
  Accent1: string, // Accent1 theme color
  Accent2: string, // Accent2 theme color
  Accent3: string, // Accent3 theme color
  Accent4: string, // Accent4 theme color
  Accent5: string, // Accent5 theme color
  Accent6: string, // Accent6 theme color
  MajorFontFamily: string, // Major text theme font family
  MinorFontFamily: string, // Minor text theme font family
}
```

GET THEME CONTENT

Parameters	Return Value	Response Model
getThemeContent(options: **GetThemeContentOptions**)	Promise	ThemeModel

GET THEME CONTENT OPTIONS OBJECT

```
type GetThemeContentOptions = {
  id: string, // theme id
}
```

THEME MODEL OBJECT

```
type ThemeModel = {
  Colors: {
    Dark1: string, // Dark1 theme color
    Dark2: string, // Dark2 theme color
    Light1: string, // Light1 theme color
    Light2: string, // Light2 theme color
    Accent1: string, // Accent1 theme color
    Accent2: string, // Accent2 theme color
    Accent3: string, // Accent3 theme color
    Accent4: string, // Accent4 theme color
    Accent5: string, // Accent5 theme color
    Accent6: string, // Accent6 theme color
    Hyperlink: string, // Hyperlink theme color
    HyperlinkFollowed: string, // Followed hyperlink theme color
  },
  Fonts: {
    MajorFont: ThemeFont,
    MinorFont: ThemeFont,
  }
  Images: Array<ThemeImage>,
  Constants: Array<ThemeConstant>,
}
type ThemeFont = {
  Family: string, // font family
  Style: string, // font style
  Size: string, // font size
  Weight: string, // font weight
}
type ThemeImage = {
  Name: string, // image name
  MIMEType: string, // image MIME type
  ImageData: string, // Base64 image data
}
type ThemeConstant = {
  Key: string, // constant key
  Value: string, // constant value
}
```

Images

GET IMAGES LIST

Parameters	Return Value	Response Model
getImagesList()	Promise	Array<ImagesListItem>

IMAGES LIST ITEM OBJECT

```
type ImagesListItem = {
  _id: string, // image id
  Name: string, // image name
  MimeType: string, // image MIME type
  Thumbnail: null, // RESERVED - for now only null is supported
}
```

GET IMAGE URL

This url is used to display external image in design-time.

Parameters	Return Value
getImageUrl(options: **GetImageUrlOptions**)	imageUrl: string

GET IMAGE URL OPTIONS OBJECT

```
type GetImageUrlOptions = {
  id: string, // image id
}
```

Report Templates

GET TEMPLATES LIST

This function can be used in some custom **File View ('Web Designer API' in the on-line documentation)** implementation.

Parameters	Return Value	Response Model
getTemplatesList()	Promise	Array<TemplatesListItem>

TEMPLATES LIST ITEM OBJECT

```
type TemplatesListItem = {
  _id: string, // template id
  Name: string, // template name
}
```

GET TEMPLATE CONTENT

Parameters	Return Value	Response Model
getTemplateContent(options: **GetTemplateContentOptions**)	Promise	**Report JSON Model**

GET TEMPLATE CONTENT OPTIONS OBJECT

```
type GetTemplateContentOptions = {
```

```
  id: string, // template id
}
```

GET TEMPLATE THUMBNAIL

This function can be used in some custom **File View ('Web Designer API' in the on-line documentation)** implementation.

Parameters	Return Value	Response Model
getTemplateThumbnail(options: **GetTemplateThumbnailOptions**)	Promise	base64ImageData: string

GET TEMPLATE THUMBNAIL OPTIONS OBJECT

```
type GetTemplateThumbnailOptions = {
  id: string, // template id
}
```

Miscellaneous

CREATE RESOURCE LINK

This function creates a resource (report/image/theme) link to be stored in **RDLX** report definition. The Resource links are then resolved while rendering report. These links need to be correct for successful report preview.

Parameters	Return Value
createResourceLink(options: **ResourceLinkOptions**)	resourceLink: string

RESOURCE LINK OPTIONS OBJECT

```
type ResourceLinkOptions = {
  id: string, // resource id
  type: 'report' | 'image' | 'theme' | null, // resource type
  version: null, // RESERVED - for now only null is supported
}
```

PARSE RESOURCE LINK

The opposite operation to **createResourceLink()**.

Parameters	Return Value
parseResourceLink(resourceLink: string)	ResourceLinkOptions

UPDATE ROUTE

This function can be used to update Designer url when a new or an existing report editing is started.

Parameters	Return Value
updateRoute(options: **UpdateRouteOptions**)	void

UPDATE ROUTE OPTIONS OBJECT

```
type UpdateRouteOptions = {
  id: string, // report id
  version: null, // RESERVED - for now only null is supported
}
```

Designer Options Object

This topic describes Designer Options Object Properties. For more information see **Web Designer API (on-line documentation)**.

serverApi

Description: Specifies server-api calls for getting server resources, for example, reports, themes, data-sets, etc.

Type: DesignerServerApi

reportInfo (optional)

Description: If report **id** is specified, the corresponding report will be opened in designer when designer application is rendered.

Type:

{ id?: string }

Example:
```
designerOptions.reportInfo.id = 'MyReport.rdlx';
```

locale (optional)

Description: If the **locale** value is not specified explicitly here, the locale corresponding to the browser preferences is used.

Type: 'en' | 'zh' | 'ja'

Example:
```
designerOptions.locale = 'zh';
```

units (optional)

Description: If measurement **units** are not specified explicitly here, they are identified depending on locale.

Type: 'in' | 'cm'

Example:

```
designerOptions.units = 'cm';
```

reportItems (optional)

Description: It is possible to limit and/or reorder available report items. Specify comma-separated report items keys from following: TextBox, CheckBox, Container, Line, Shape, TableOfContents, Image, InputField, List, Table, Tablix, Chart, Bullet, Barcode, FormattedText, Sparkline, Subreport, and OverflowPlaceholder.

Type: string

Example:
```
designerOptions.reportItems = 'TextBox,CheckBox,Table,Chart,Image';
```

lockLayout

Description: By default **lockLayout** is disabled. When **lockLayout** is enabled, it is only possible to modify properties of existing report items. That is, adding a new report item or deleting an existing one is not possible as well as other operations that modify report layout structure.

Type: boolean

Example:

```
designerOptions.lockLayout = true;
```

restoreUnsavedReport

Description: By default **restoreUnsavedReport** is enabled. In this case the last unsaved report can be restored if browser tab or browser itself gets accidentally closed. When **restoreUnsavedReport** is disabled, the aforementioned functionality is not available.

Type: boolean

Example:
```
designerOptions.restoreUnsavedReport = false;
```

saveButton

Description: Specifies whether **Save** button needs to be shown. **Save** button is **not** visible by default.

Type:

{ visible: boolean }

Example:

```
designerOptions.saveButton.visible = true;
```

saveAsButton

Description: Specifies whether **Save As** button needs to be shown. **Save As** button is **not** visible by default.

Type:

{ visible: boolean }

Example:
```
designerOptions.saveAsButton.visible= true;
```

reportExplorer

Description: Specifies whether **Report Explorer** button needs to be shown. **Report Explorer** button is visible by default.

Type:

{ visible: boolean }

Example:

```
designerOptions.reportExplorer.visible= false;
```

groupEditor

Description: Specifies whether **Group Editor** button needs to be shown. **Group Editor** button is visible by default.

Type:

{ visible: boolean }

Example:
```
designerOptions.groupEditor.visible = false;
```

toolBox

Description: Specifies whether left-side menu **Toolbox** needs to be shown. **Toolbox** is visible by default.

Type:

{ visible: boolean }

Example:
```
designerOptions.toolBox.visible = false;
```

insertTab

Description: Specifies whether application bar menu **Insert Tab** needs to be shown. **Insert Tab** is not visible by default. **Toolbox** and **Insert Tab** are interchangeable.

Type:

{ visible: boolean }

Example:

```
designerOptions.insertTab.visible = true;
```

propertiesTab

Description:

- **visible** - Specifies whether **Properties Tab** needs to be shown. **Properties Tab** is visible by default.
- **mode -** Specifies available properties modes. The default value is 'Both'.
- **dataSets** -
    - visible - Specifies whether Data Sets section needs to be shown. 'Data Sets' section is visible by default.
    - canModify - Specifies whether it is possible to modify (including add/remove) data sets. By default this feature is disabled.

- **defaultMode** - Relevant only when mode is 'Both'. If undefined, the last used properties mode is set.

Type:
```
{
  visible: boolean,
  mode: 'Both' | 'Advanced' | 'Basic',
  defaultMode?: 'Advanced' | 'Basic',
}
```

Example:

```
designerOptions.propertiesTab.visible = false;
designerOptions.propertiesTab.mode = 'Basic';
designerOptions.propertiesTab.defaultMode = 'Advanced';
```

dataTab

Description:

- **visible** - Specifies whether **Data Tab** needs to be shown. **Data Tab** is visible by default.
- **dataSources -**
  - visible - Specifies whether **Data Sources** section needs to be shown. **Data Sources** section is visible by default.
- **dataSets** -
  - visible - Specifies whether **Data Sets** section needs to be shown. **Data Sets** section is visible by default.
  - canModify - Specifies whether it is possible to modify (including add/remove) data sets. By default this feature is disabled.

- **parameters** -
  - visible - Specifies whether **Parameters** section needs to be shown. **Parameters** section is visible by default.
  - canModify - Specifies whether it is possible to modify (including add/remove) report parameters. By default this feature is enabled.

- **commonValues** -
  - visible - Specifies whether **Common Values** section needs to be shown. **Common Values** section is visible by default.

Type:

```
{
 visible: boolean,
 dataSources: {
  visible: boolean,
 },
 dataSets: {
  visible: boolean,
  canModify: boolean,
 },
 parameters: {
  visible: boolean,
  canModify: boolean,
 },
 commonValues: {
  visible: boolean,
 },
}
```

Example:

```
designerOptions.dataTab.visible = false;
designerOptions.dataTab.dataSources.visible = false;
designerOptions.dataTab.dataSets.visible = false;
designerOptions.dataTab.dataSets.canModify = true;
designerOptions.dataTab.parameters.visible = false;
designerOptions.dataTab.parameters.canModify = false;
```

```
designerOptions.dataTab.dataSources.visible = false;
```

gridSize

Description:

- visible - Specifies whether **Grid Size** editor in **Status Bar** needs to be shown. **Grid Size** editor is visible by default.
- value - If **Grid Size** editor is not visible, it is possible to specify grid size value in **in/cm** (inches or centimeters).

Type:
{
visible: boolean,
value?: string,
}

Example:

```
designerOptions.gridSize.visible = false;
designerOptions.gridSize.value = '0.75cm';
/* or */
designerOptions.gridSize.value = '0.5in';
```

showGrid

Description:

- visible - Specifies whether **Show Grid** toggle in **Status Bar** needs to be shown. **Show Grid** toggle is visible by default.
- value - If **Show Grid** toggle is not visible, it is possible to specify show grid value as *true* or *false*.

Type:
{
visible: boolean,
value?: boolean,
}

Example:

```
designerOptions.showGrid.visible = false;
designerOptions.showGrid.value = false;
```

canEditDataSource

Description: Indicates whether a data source can be edited.

Type: boolean

Example:

```
@param options CanEditDataSourceOptions object
```

canEditDataSet

Description: Indicates whether a data set can be edited.

Type: boolean

Example:

```
@param options CanEditDataSetOptions object
```

canAddDataSetForDataSource

Description: Returns a data set initialized for the specified data source.

Type: boolean

Example:

```
@param options InitDataSetForDataSourceOptions object
```

initDataSetForDataSource

Description: Returns properties available for the specified data set.

Type: Dataset

Example:

```
@param options GetDataSetPropertiesOptions object
```

getDataSetSchema

Description: Gets schema for the specified data set.

Example:

```
@param option GetDataSetSchemaOptions object
getDataSetSchema: (options: GetDataSetSchemaOptions) => Promise;
```

fonts

Description: Specifies the list of fonts displayed in font properties drop-downs all over Designer. If **fonts** are not specified explicitly here, the default list of fonts is used.

Type: Array<string>

Example:

```
designerOptions.fonts = ['Arial', 'Courier New', 'Times New Roman'];
```

openViewer

Description: You can plug-in **Report Viewer** component by providing **openViewer()** function implementation to **DesignerOptions** object. When **openViewer()** is implemented and passed to **DesignerOptions**, **Preview** button appears in Designer application bar.

Example:

```
designerOptions.openViewer = openViewerImpl;
```

openFileView

Description: You can plug-in **File View** component by providing **openFileView()** function implementation to **DesignerOptions** object. When **openFileView()** is implemented and passed to **DesignerOptions**, **File** tab appears in Designer application bar.

Example:

```
designerOptions.openFileView = openFileViewImpl;
```

onSave

Description: You can specify behavior for **Save Report** scenario by providing **onSave()** function implementation to **DesignerOptions** object.

Example:

```
designerOptions.onSave = onSaveImpl;
```

onSaveAs

Description: You can specify behavior for **Save Report As** scenario by providing **onSaveAs()** function implementation to **DesignerOptions** object.

Example:

```
designerOptions.onSaveAs = onSaveImpl;
```

onOpen

Description: You can specify behavior for **Open Report** scenario by providing **onOpen()** function implementation to **DesignerOptions** object.

Example:

```
designerOptions.onOpen = onOpenImpl;
```

openDataSourceEditor

Description: You can specify behavior for **Add/Edit Data Source** scenario by providing **openDataSourceEditor()** function implementation to **DesignerOptions** object.

Example:

```
designerOptions.openDataSourceEditor = openDataSourceEditorImpl;
```

openButton

Description: Specifies whether **Open** button needs to be shown. **Open** button is **not visible** by default.

Type: { visible: boolean }

Example:

```
designerOptions.openButton.visible = true;
```

dataSetPicker

Description:

- **mode**- Data Set Picker mode.
  open- Specifies behavior on opening Data Set Picker.
- **close**- Specifies behavior on closing Data Set Picker

Example:

```
designerOptions.dataSetPicker.mode = 'Panel';
designerOptions.dataSetPicker.open = openDataSetPickerImpl;
```

```
designerOptions.dataSetPicker.close = closeDataSetPickerImpl;
```

reportItemsFeatures

Barcode features

defaultSymbology

Description: Overrides the default symbology used for newly-created barcodes. By default new barcodes are created with QR Code symbology.

Type: BarcodeSymbology

Example:

```
designerOptions.reportItemsFeatures.barcode.defaultSymbology = 'Code_128_A';
```

symbologies

Description: Limits the list of barcode symbologies available for creation. By default all barcode symbologies supported by ActiveReports are available..

Type: Array<BarcodeSymbology>

Example:

```
designerOptions.reportItemsFeatures.barcode.symbologies = ['Code_128_A', 'Code_128_B',
'Code_128_C']
```

Table features

autoFillHeader

Description: Specifies whether **Table Header** needs to be auto-filled when a field is dropped to **Table Details**. For example, if ProductName field is dropped to Details, Product Name value is set to **Header**. By default this feature is enabled.

Type: Boolean

Example:

```
designerOptions.reportItemsFeatures.table.autoFillHeader = false;
```

autoFillFooter

Description: Specifies whether **Table Footer** needs to be auto-filled when a field is dropped to **Table Details**. For example, if ProductName field is dropped to Details, =Count([ProductName]) value is set to **Footer**. By default this feature is disabled.

Type: Boolean

Example:

```
designerOptions.reportItemsFeatures.table.autoFillFooter = true;
```

canMergeCellsVertically

Description: Specifies whether vertical merge of cells is enabled within **Table Header**, **Details**, and **Footer**. By default this

feature is enabled.

Type: Boolean

Example:

```
designerOptions.reportItemsFeatures.table.canMergeCellsVertically = false;
```

Tablix features

autoFillCorner

Description: Specifies whether **Tablix Corner Cell** needs to be auto-filled when a field is dropped to **Tablix Row Group Cell**. For example, if ProductName field is dropped to Row Group Cell, Product Name value is set to Corner Cell. By default this feature is enabled.

Type: Boolean

Example:

```
designerOptions.reportItemsFeatures.tablix.autoFillCorner = false;
```

canUseWizard

Description: Specifies whether Tablix Wizard is available for creating or editing Tablix. By default this feature is enabled.

Type: Boolean

Example:

```
designerOptions.reportItemsFeatures.tablix.canUseWizard = false;
```

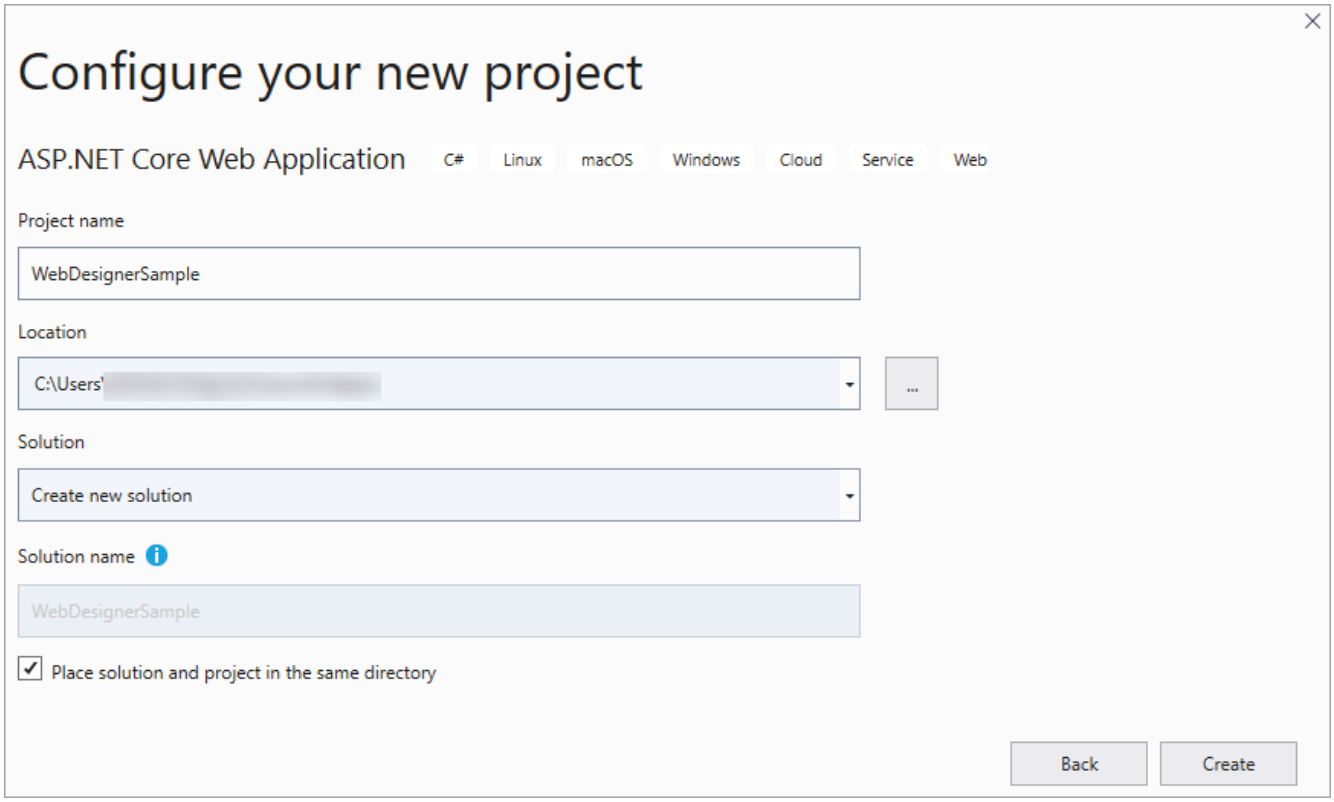
Create a Simple Web Designer Sample

The topic describes creating Web Designer sample using ASP .NET MVC Core and ASP .NET MVC.

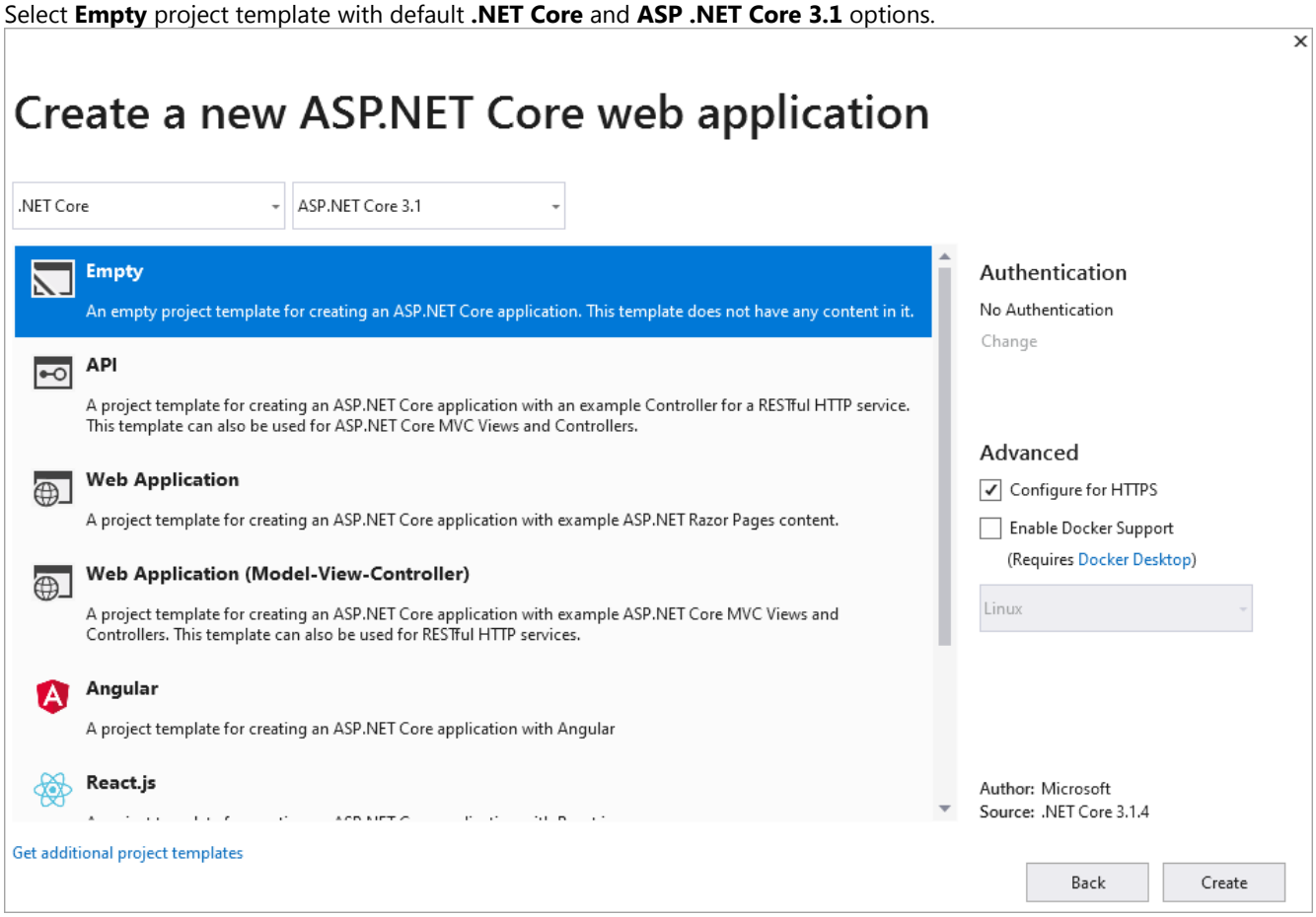
ASP .NET MVC Core

The steps to create a Web Designer sample using ASP .NET MVC Core application are as follows:

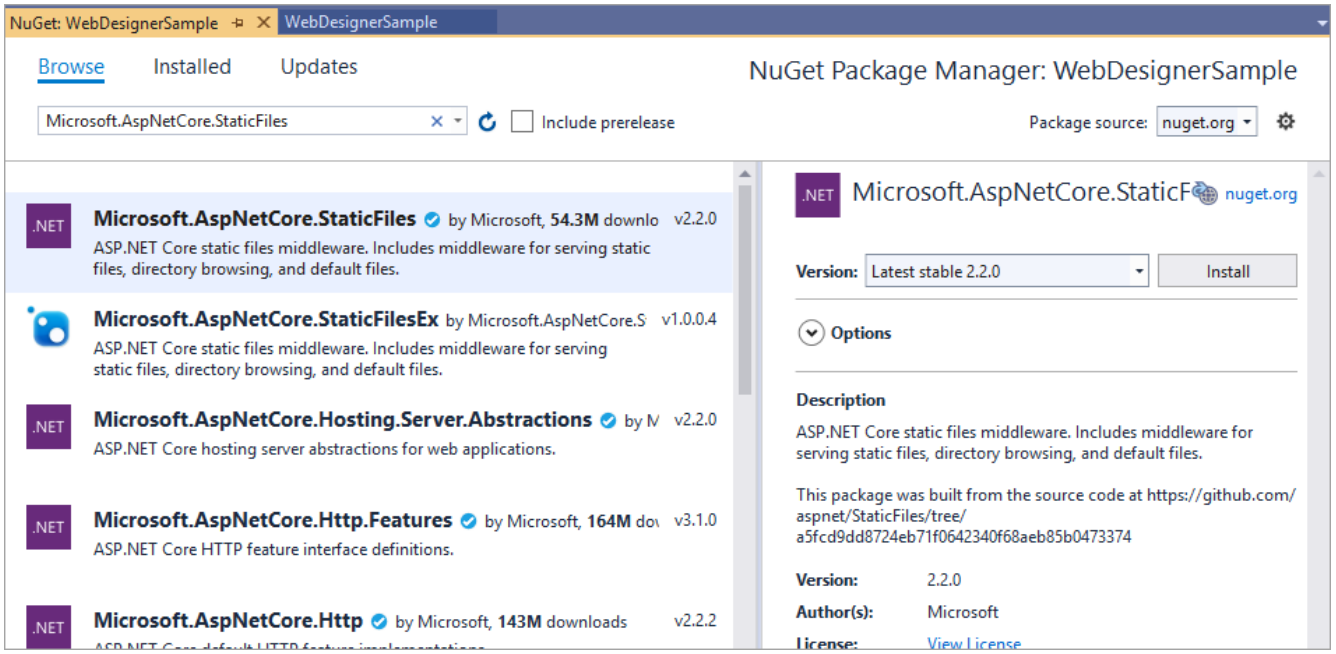
1. Open **Microsoft Visual Studio 2019** and create a new **ASP .NET Core Web Application** project.



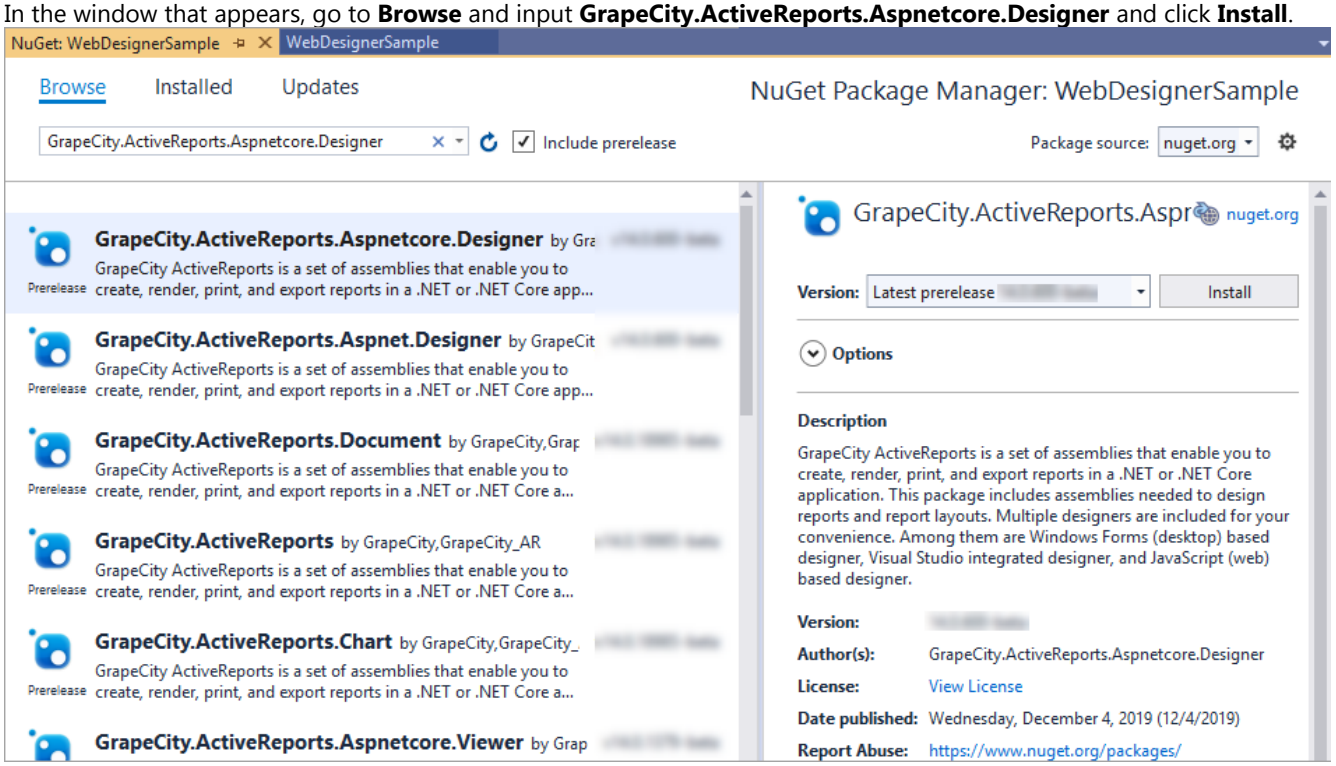
2. Select **Empty** project template with default **.NET Core** and **ASP .NET Core 3.1** options.



3. In Solution Explorer right-click **Dependencies** and go to **Manage NuGet Packages**.
4. In the window that appears, go to **Browse** and input **Microsoft.AspNetCore.StaticFiles**, select the latest version and click **Install**.



5. In the window that appears, go to **Browse** and input **GrapeCity.ActiveReports.Aspnetcore.Designer** and click **Install**.



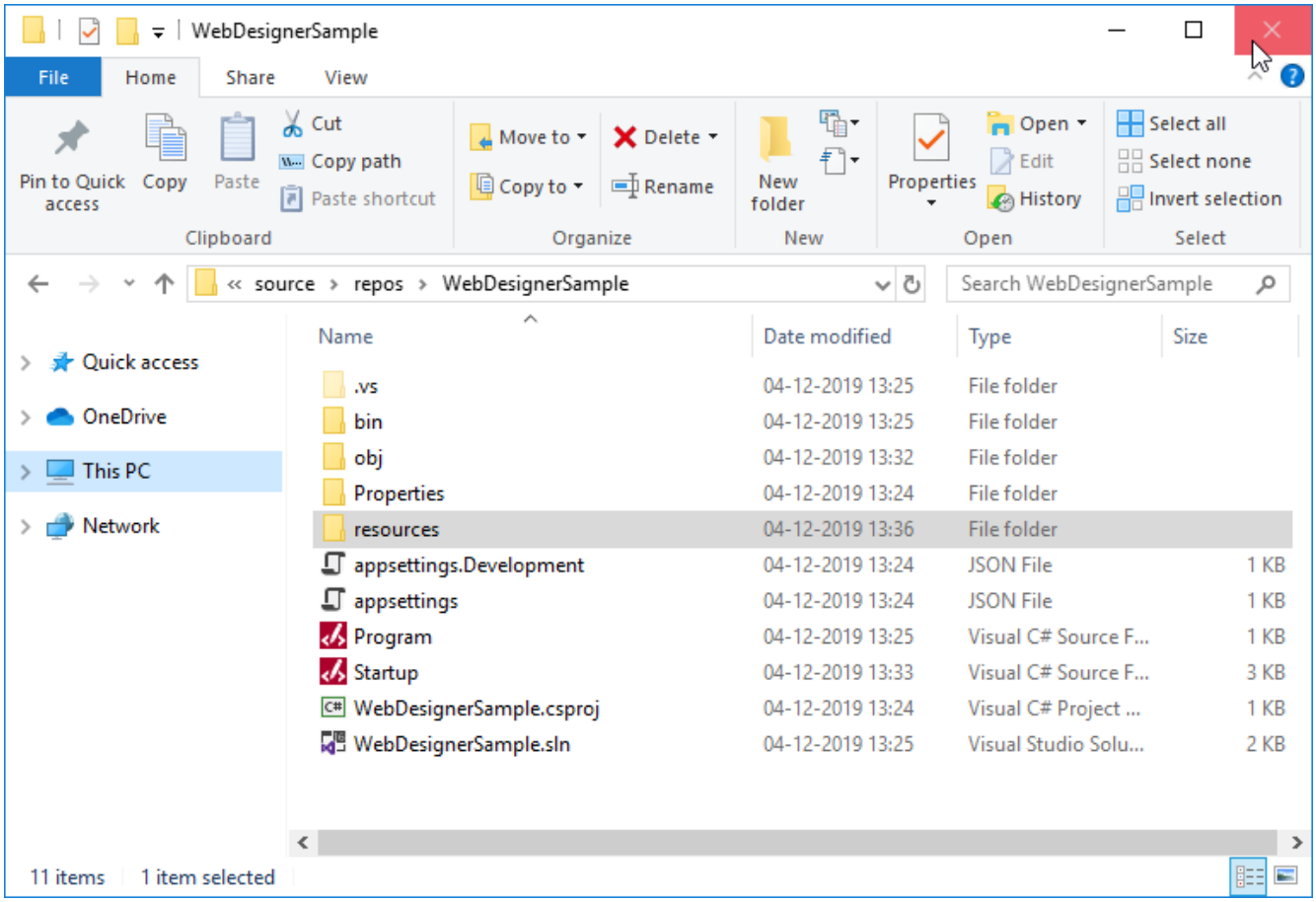
6. In the **License Acceptance** dialog that appears, click **I Accept**.
7. In Solution Explorer, find **Startup.cs** and modify its content as follows:

Startup.cs

```
using System.IO;
using Microsoft.AspNetCore.Builder;
```

```
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.DependencyInjection;
using GrapeCity.ActiveReports.Aspnetcore.Designer;
namespace WebDesignerSample
{
        public class Startup
        {
                // resources (reports, themes, images) location
                private static readonly DirectoryInfo ResourcesRootDirectory = new
DirectoryInfo(".\\resources\\");
                public void ConfigureServices(IServiceCollection services)
                {
                        // web designer services
                        services.AddDesigner();
                }
                public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
                {
                        // web designer middleware
                        app.UseDesigner(config =>
config.UseFileStore(ResourcesRootDirectory));
                        // static files middlewares
                        app.UseDefaultFiles();
                        app.UseStaticFiles();
                }
        }
}
```

8. Create 'resources' folder in your sample project root; you can put your existing reports, themes, and images in this folder.



9. To use npm packages, your project must contain **package.json** file. Run the following command in the command line before installing any code dependencies:

```
npm init -y
```

10. Download and install the WebDesigner-related files and folders from NPM using the following command in the command line:

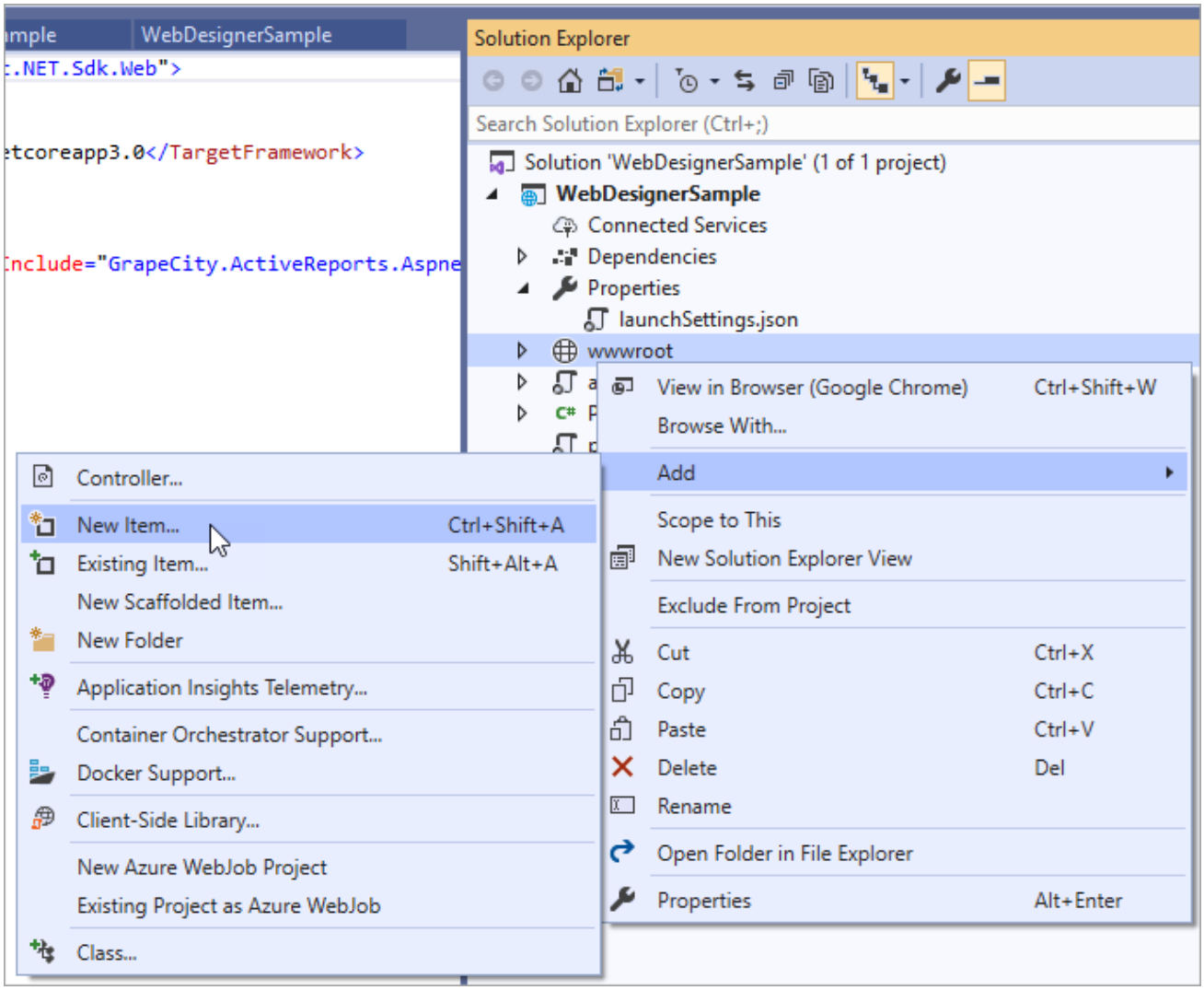
```
npm install @grapecity/ar-designer
```

The designer files/folders will be downloaded in your current directory: .\node_modules\@grapecity\ar-designer\dist

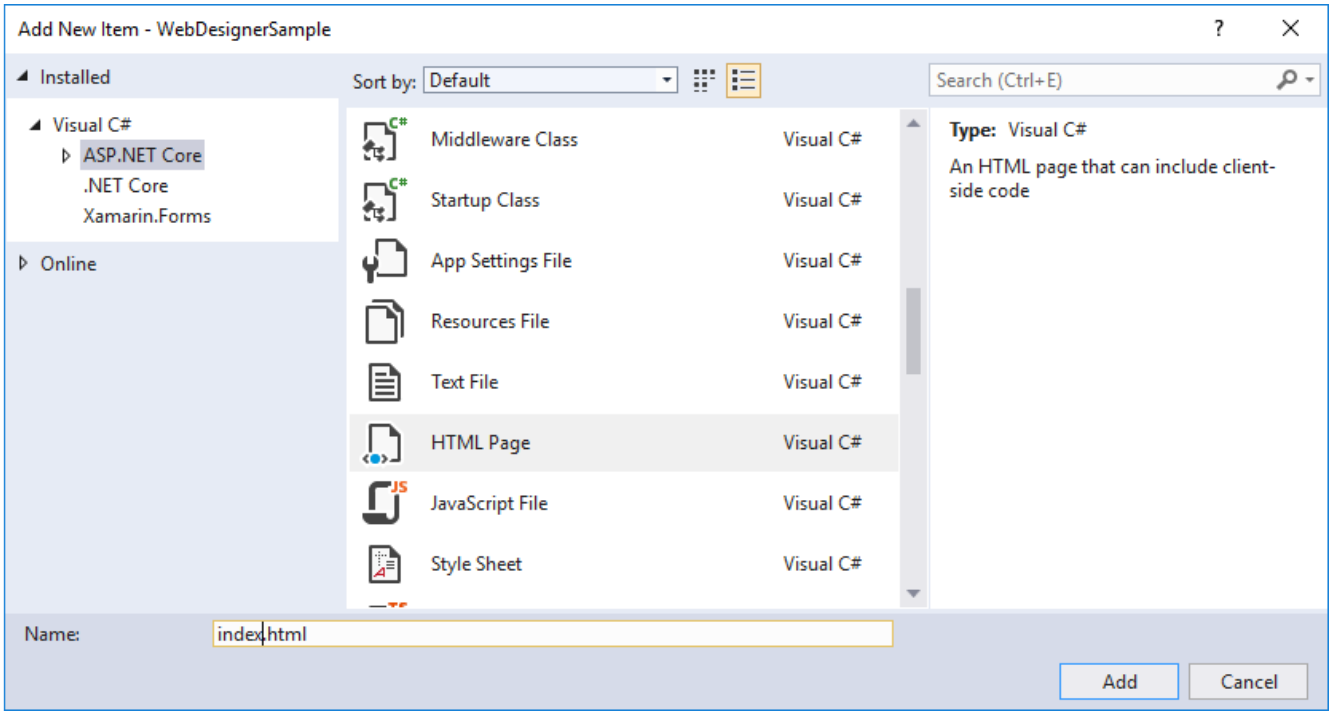
11. Copy the following designer files/folder and paste it to your sample project **wwwroot** subfolder:
    - baseServerApi.js
    - web-designer.css
    - web-designer.js
    - vendor folder

Optionally you can also copy **file-dialog.css** and **file-dialog.js** if you would like to use our sample dialog for saving reports.

12. In Solution Explorer, right-click wwwroot and select **Add** > **New Item**.



13. Select **HTML Page** item type, input **index.html** and click **Add**.



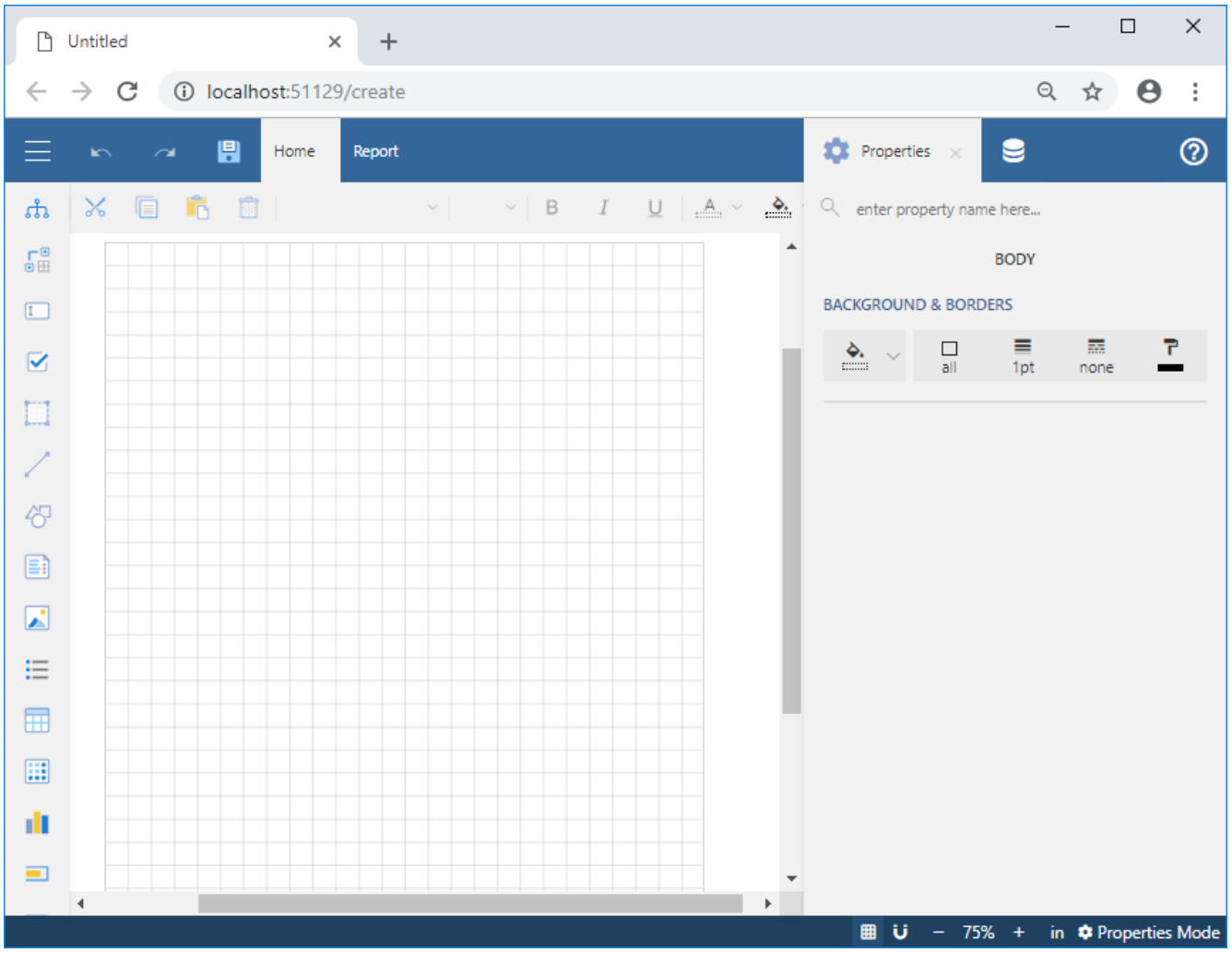
14. In Solution Explorer, find newly-added **index.html** and modify its content as follows:

index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Web Designer Sample</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta http-equiv="x-ua-compatible" content="ie=edge">
    <!-- No Virtual Directory -->
    <base href="/">
    <!-- designer-related css -->
    <link rel="stylesheet" href="vendor/css/materialdesignicons.min.css" media="all"
type="text/css" />
    <link rel="stylesheet" href="vendor/css/bootstrap.min.css" />
    <link rel="stylesheet" href="vendor/css/font-awesome.min.css">
    <link rel="stylesheet" href="vendor/css/ionicons.min.css">
    <link rel="stylesheet" href="vendor/css/fonts-googleapis.css" type="text/css">
    <link rel="stylesheet" href="web-designer.css" />
</head>
<body class="theme-blue">
    <!-- designer-related js -->
    <script src="vendor/js/jquery.min.js"></script>
    <script src="vendor/js/bootstrap.min.js"></script>
    <script src="baseServerApi.js"></script>
    <script src="web-designer.js"></script>
    <!-- designer root div -->
    <div id="designer-id" style="width: 100%; height: 100%;"></div>
    <script>
```

```
        // create designer options
        var designerOptions =
GrapeCity.ActiveReports.WebDesigner.createDesignerOptions(baseServerApi);
        // render designer application
        GrapeCity.ActiveReports.WebDesigner.renderApplication('designer-id',
designerOptions);
    </script>
</body>
</html>
```

15. Build your solution (Build> Build Solution) and run it. WebDesigner with a blank RDL report opens in your browser.



16. If you would like to open not a blank report but one of your existing reports in resources subfolder (added above in step 12), you will need to add the following line with your report name in index.html after **createDesignerOptions()** function call:

index.html

```
designerOptions.reportInfo.id = "MyReport.rdlx";
```

17. In case you copied **file-dialog.css** and **file-dialog.js** to the sample project wwwroot subfolder in step 14., you can plug-in our sample dialog for saving reports.

    Following steps are required to be performed in index.html to plug-in the dialog component:

i. In **<head>** tag, add file-dialog.css near **web-designer.css**:

```
<link rel="stylesheet" href="file-dialog.css" />
<link rel="stylesheet" href="web-designer.css" />
```

ii. In **<body>** tag, add file-dialog.js near **web-designer.js**:

```
<script src="file-dialog.js"></script>
<script src="web-designer.js"></script>
```

iii. Near **designer root div** and **dialog root div**:

```
<!-- designer root div -->
< div id="designer-id" style="width: 100%; height: 100%;"></div>
<!-- save as dialog root div -->
< div id="save-as-dialog-id" style="position: absolute; top: 0; left: 0; width: 100%;
height: 100%; display: none; z-index: 9999;"></div>
```

iv. Modify **<script>** tag contents where designer application is rendered:

```
<script>
    var showElement = function (id) {
        if (!id) return;
        ($('#' + id)).css('display', 'block');
    };
    var hideElement = function (id) {
        if (!id) return;
        ($('#' + id)).css('display', 'none');
    };
    var designerId = 'designer-id';
    var saveAsDialogId = 'save-as-dialog-id';
    function onSaveAs(options) {
        showElement(saveAsDialogId);
        // render save-as dialog
        fileDialog.createSaveReportAsDialog(saveAsDialogId, {
            locale: options.locale,
            api: {
                getReportsList: function () {
                    return baseServerApi.getReportsList()
                        .then(function (reportsList) {
                            return reportsList.map(function (reportInfo) {
                                return { path: reportInfo.Name };
                            });
                        });
                },
                saveReport: function (saveOptions) {
                    return baseServerApi.saveNewReport({
                        name: saveOptions.path,
                        content: options.reportInfo.content,
                    }).then(function (saveResult) {
```

```
                        return { id: saveResult.Id };
                    });
                },
            },
            reportInfo: {
                path: options.reportInfo.name,
            },
            onSuccess: function (saveResult) {
                hideElement(saveAsDialogId);
                options.onSuccess({ id: saveResult.id, name: saveResult.path });
            },
            onClose: function () {
                hideElement(saveAsDialogId);
            },
        });
    };
    // create designer options
    var designerOptions =
GrapeCity.ActiveReports.WebDesigner.createDesignerOptions(baseServerApi);
    // enable showing save-as button
    designerOptions.saveAsButton.visible = true;
    // specify behavior on save-as
    designerOptions.onSaveAs = onSaveAs;
    // render designer application
    GrapeCity.ActiveReports.WebDesigner.renderApplication(designerId, designerOptions);
</script>
```

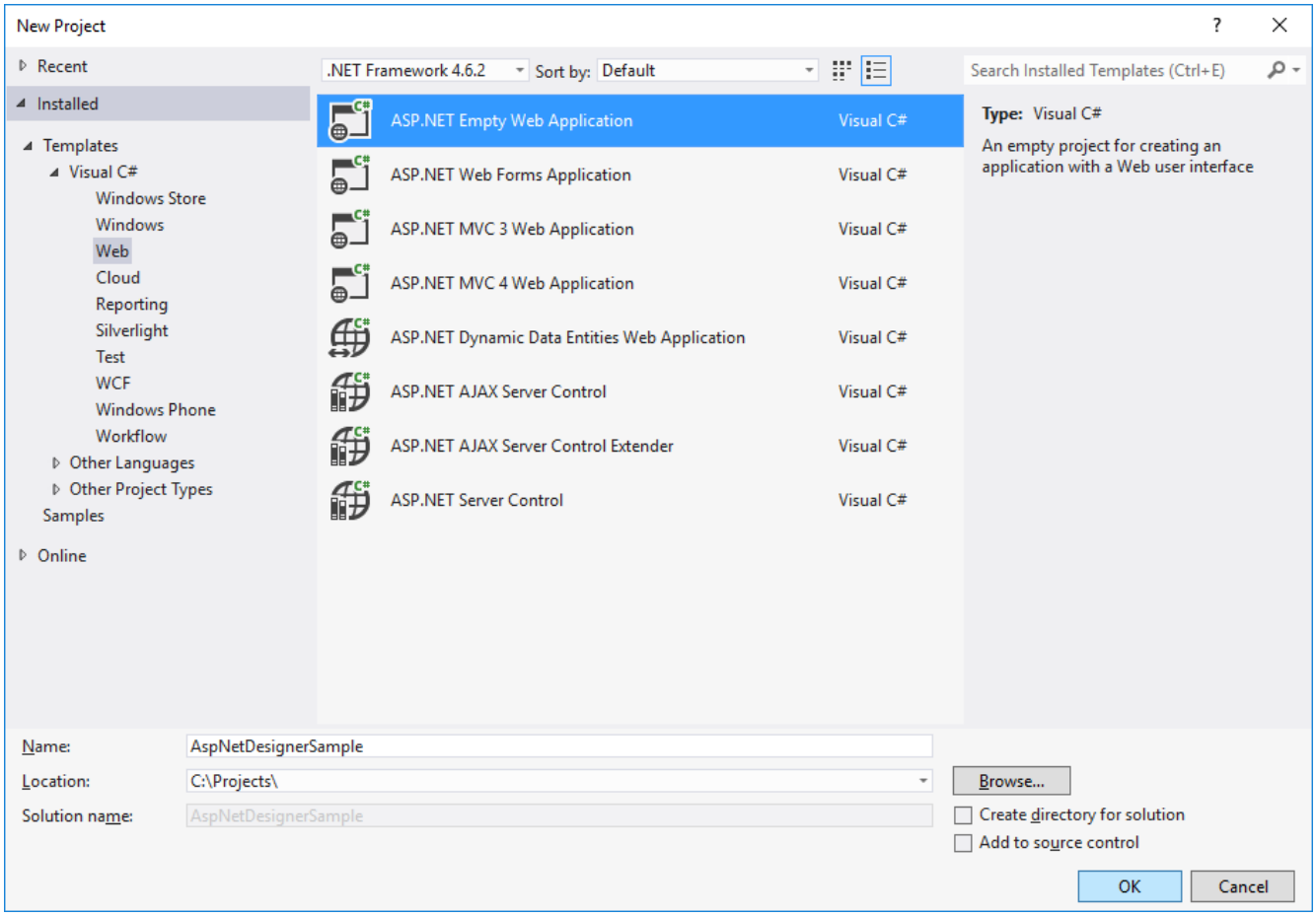
You get a designer with plugged-in sample save-as dialog. Such minimalistic designer can be used for editing the existing reports without adding new data sets.

However, in case you need to create brand-new reports, add data sets and preview reports from designer, please refer **Web Designer API (on-line documentation)** topic.

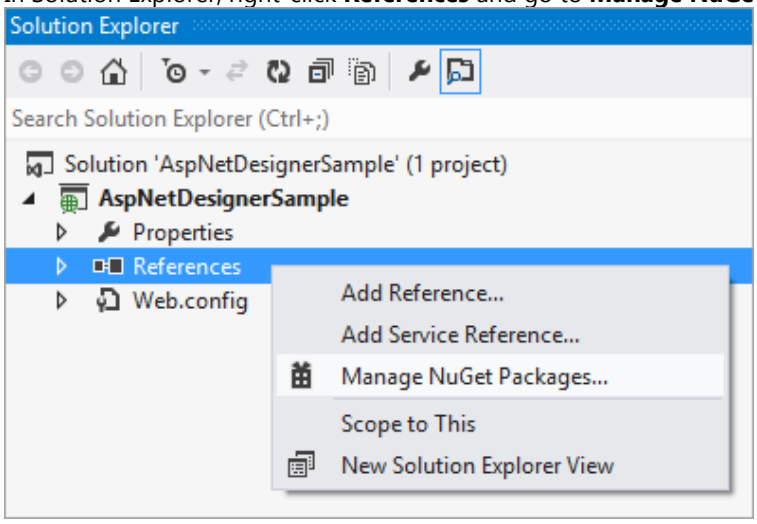
ASP .NET MVC

The steps to create a Web Designer sample using ASP .NET MVC application are as follows:

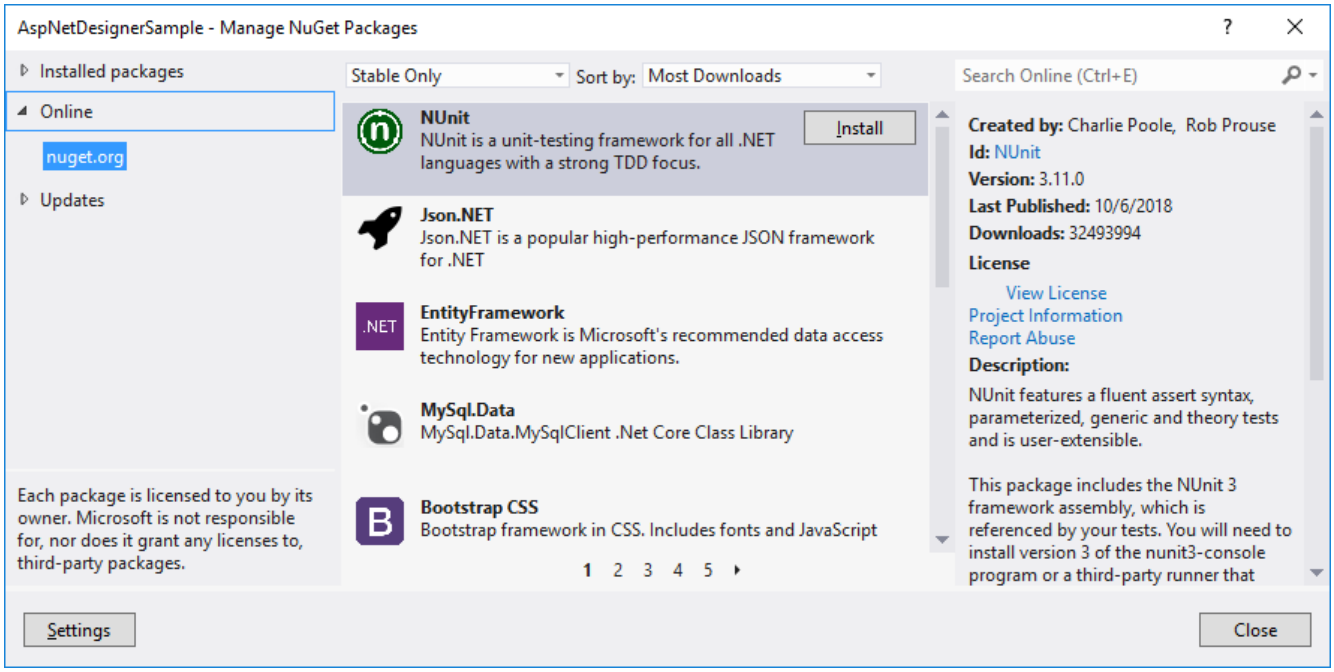
1. Open **Microsoft Visual Studio 2013** and create a **NET Framework 4.6.2 ASP .NET Empty Web Application** project.



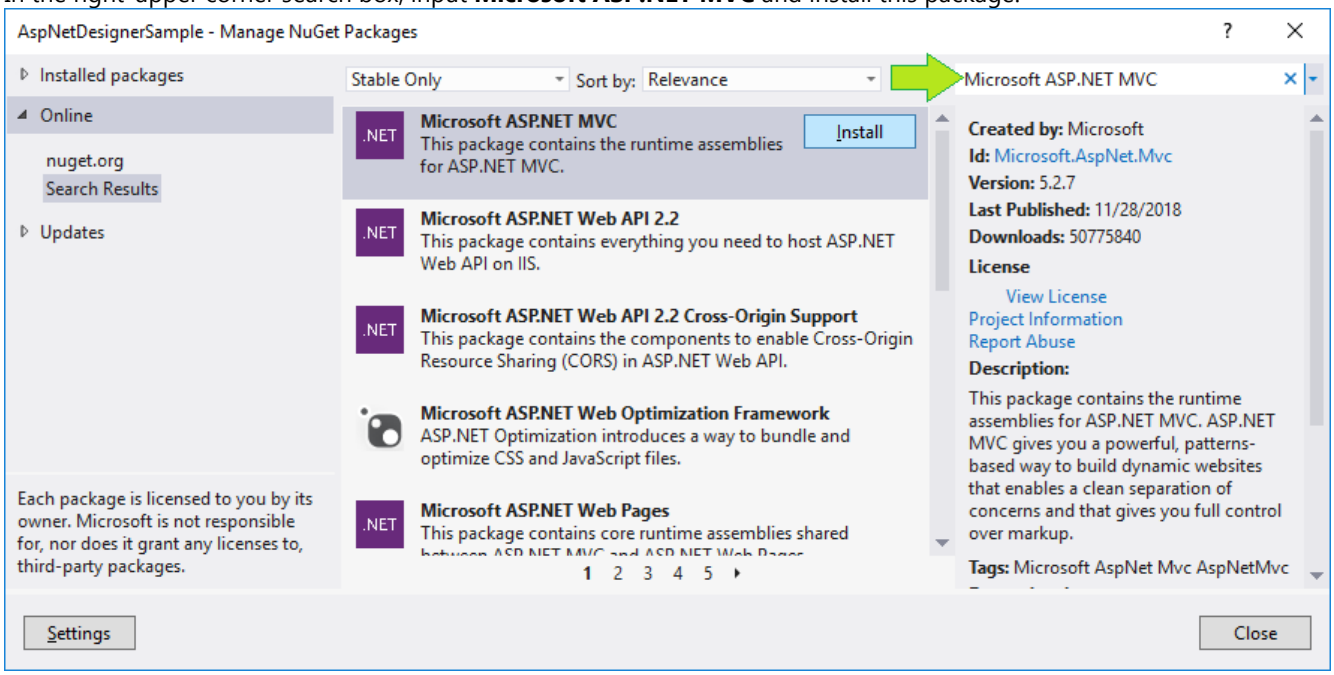
2. In Solution Explorer, right-click **References** and go to **Manage NuGet Packages**.



3. In the window that appears, go to **Online > nuget.org**.

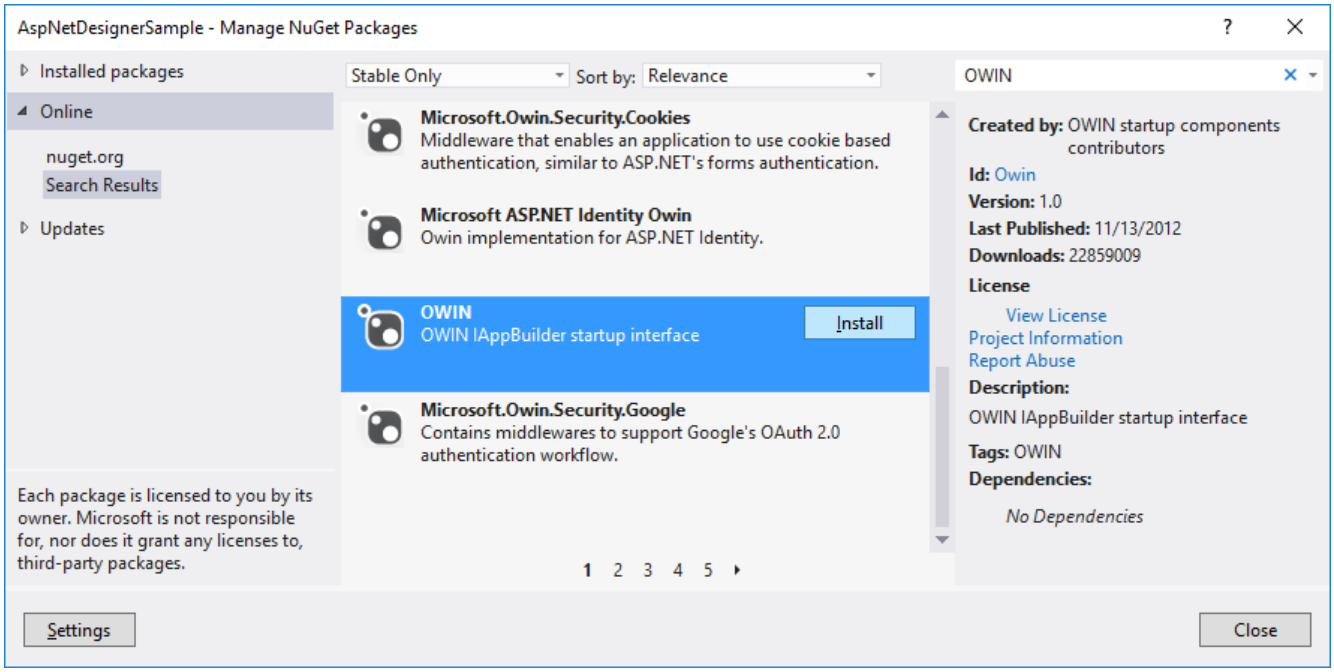


4. In the right-upper corner search box, input **Microsoft ASP.NET MVC** and install this package.

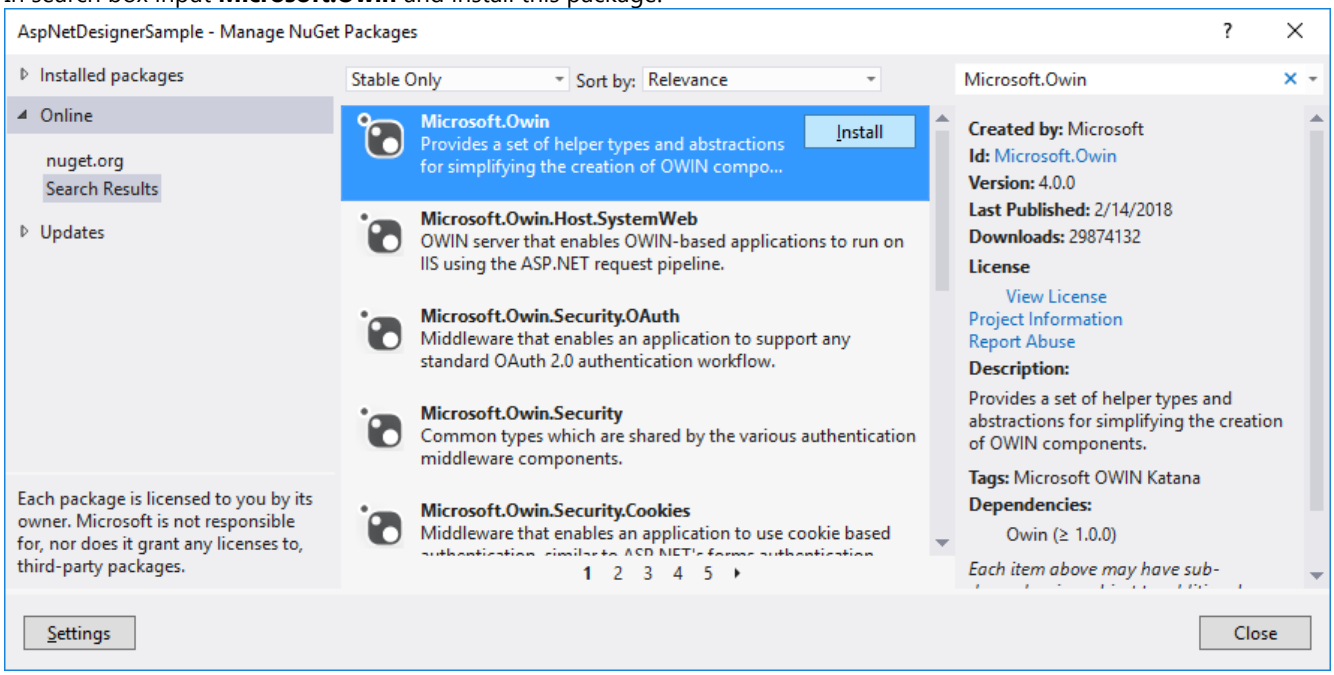


In the **License Acceptance** dialog that appears, click **I Accept.**

5. In search box input **OWIN**, find 'OWIN' in search results and install this package:

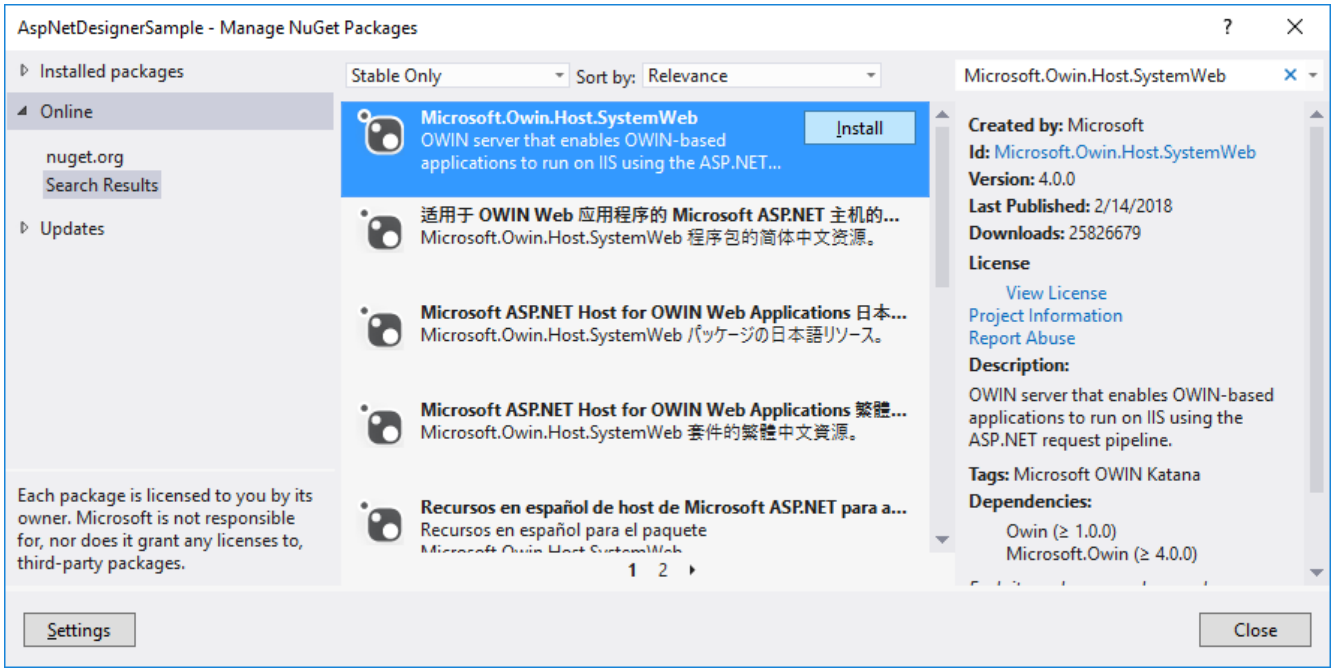


6. In search box input **Microsoft.Owin** and install this package.



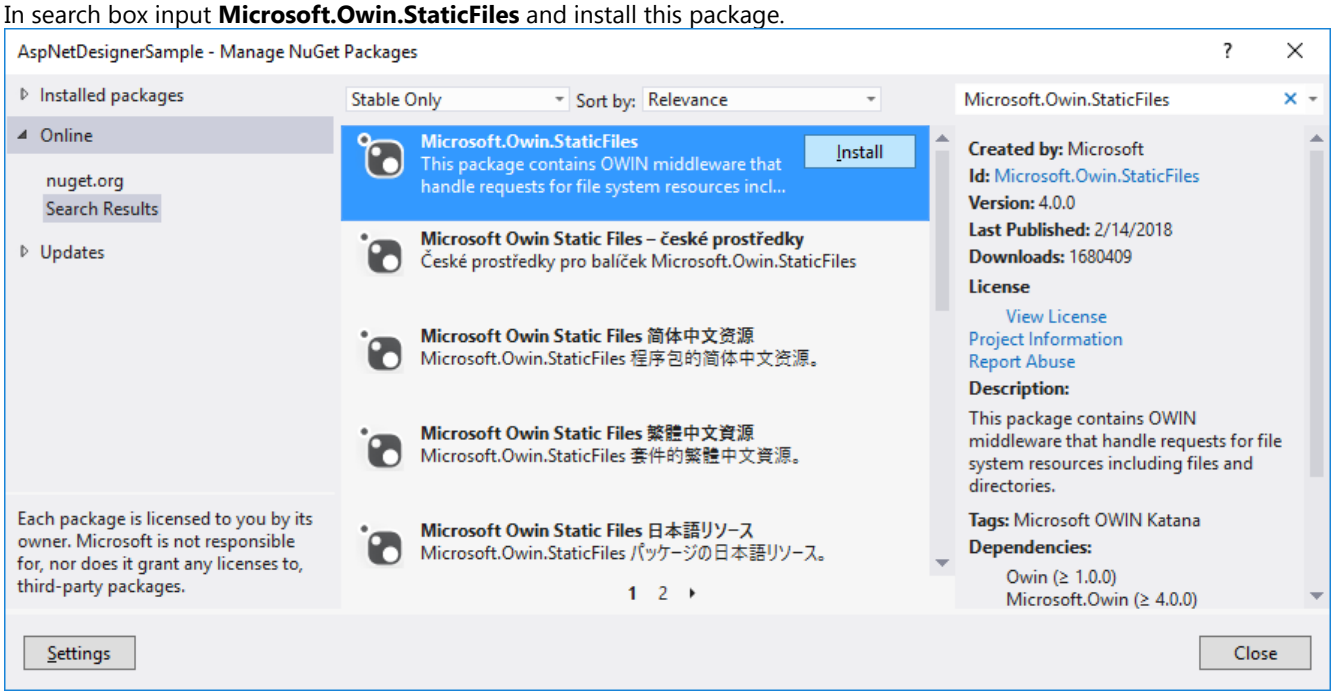
In the **License Acceptance** dialog that appears, click **I Accept.**

7. In search box input **Microsoft.Owin.Host.SystemWeb** and install this package.



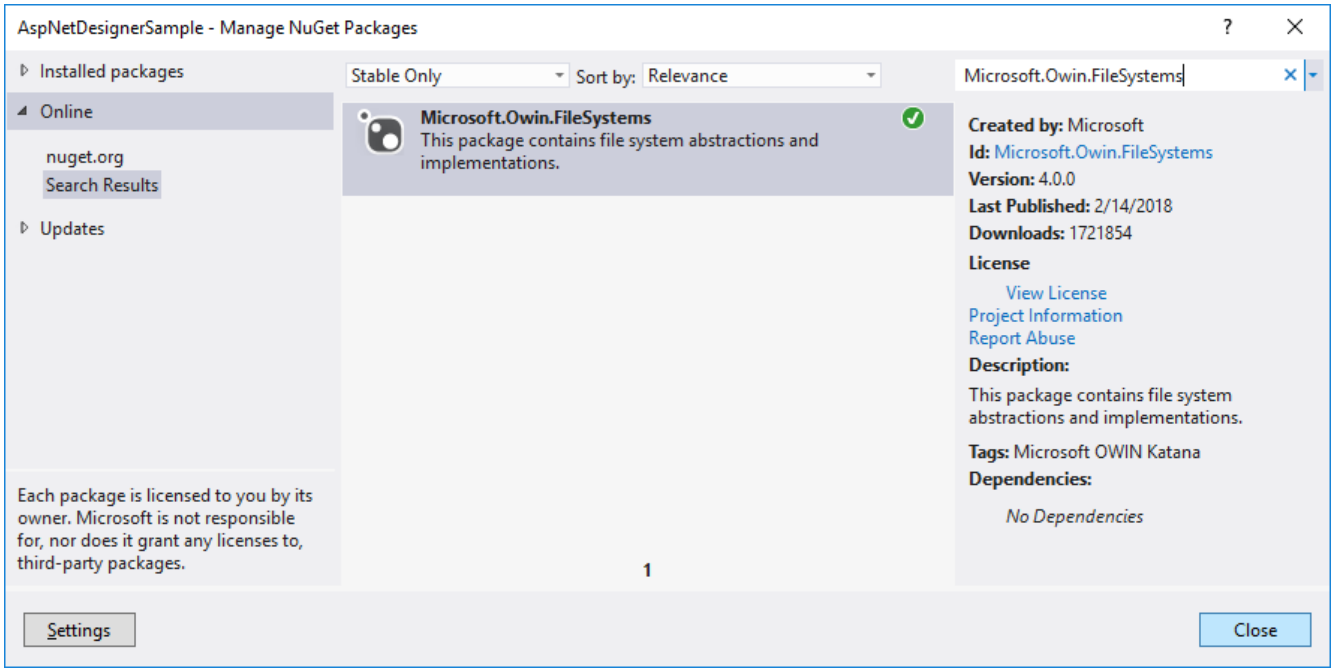
In the appeared **License Acceptance** dialog click **I Accept**.

8. In search box input **Microsoft.Owin.StaticFiles** and install this package.



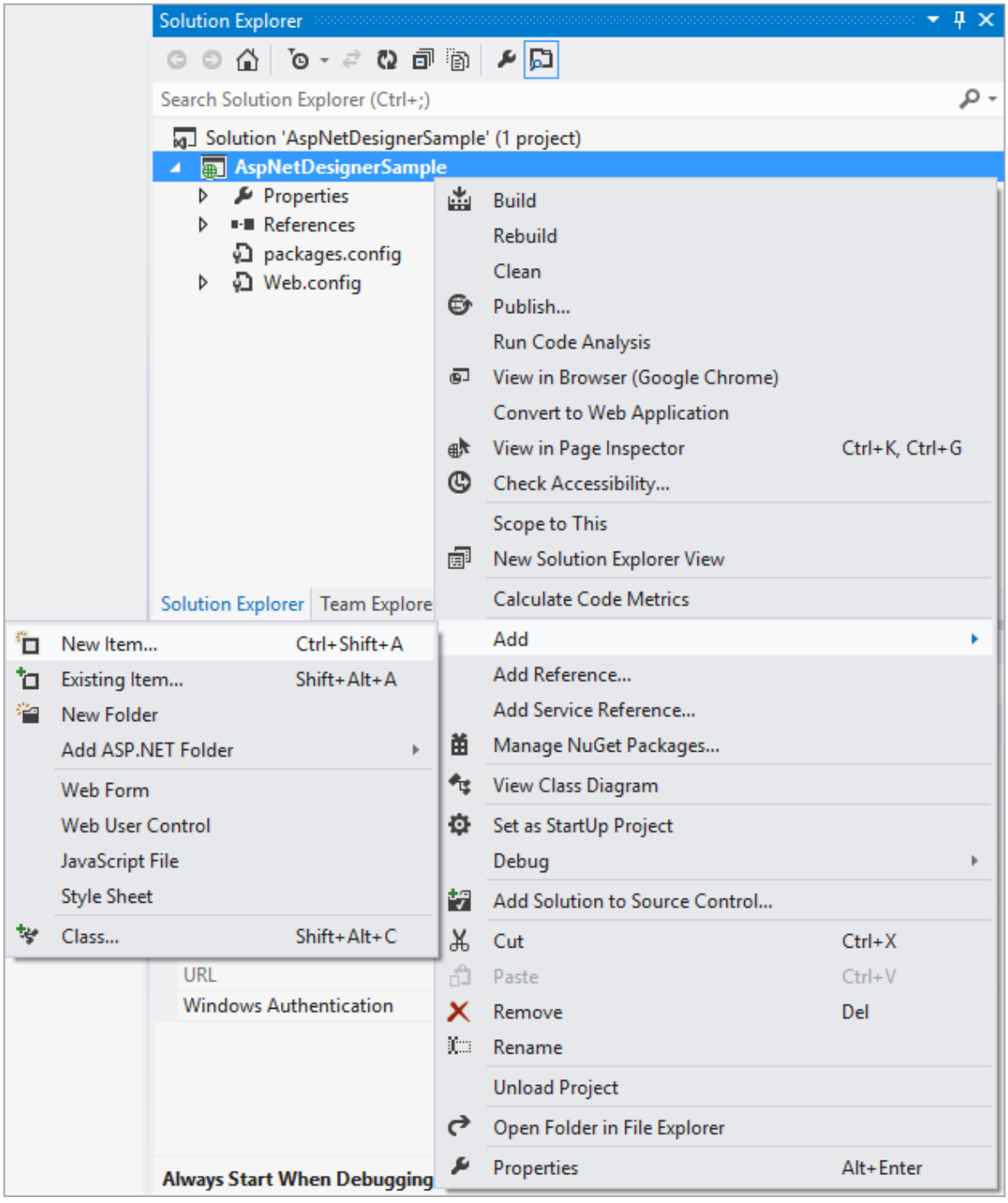
In the **License Acceptance** dialog that appears, click **I Accept.**

9. In search box input **Microsoft.Owin.FileSystems**.
However, this package should be already installed together with **Microsoft.Owin.StaticFiles** in the previous step.

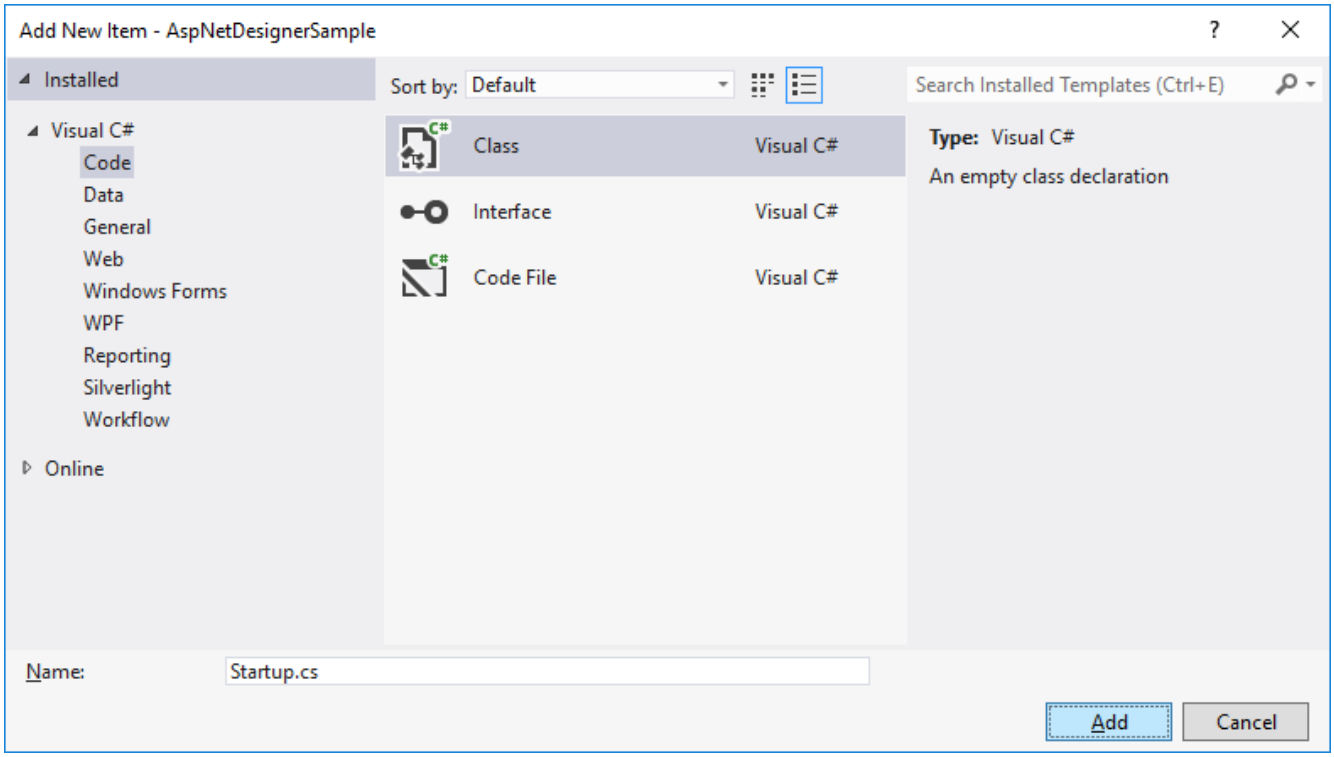


Therefore, all the needed packages are installed. Now, lets add the designer package in the next step.

10. In the search box in NuGet Package Manager, input **GrapeCity.ActiveReports.Aspnet.Designer** and click **Install**.
11. In the **Preview Changes** dialog that appears, click **OK**.
12. In the **License Acceptance** dialog that appears, click **I Accept**.
13. In Solution Explorer, right-click your project, go to **Add > New Item...**



In the window that appears, go to **Code > Class**, input **Startup.cs** and click **Add**.

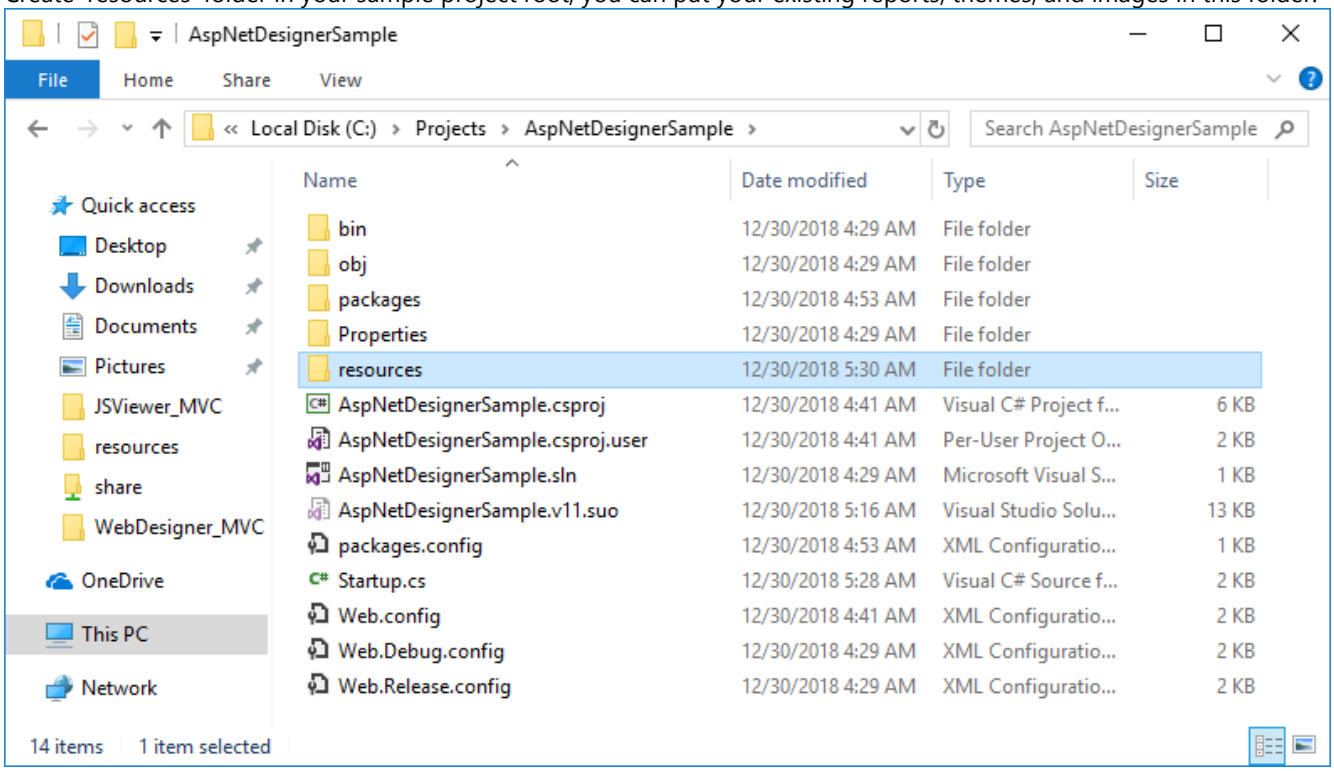


Modify the contents of newly-added **Startup.cs** as follows:

```
using System;
using System.IO;
using System.Linq;
using System.Web;
using GrapeCity.ActiveReports.Aspnet.Designer;
using Owin;
using Microsoft.Owin;
using Microsoft.Owin.StaticFiles;
using Microsoft.Owin.FileSystems;
using System.Web.Mvc;
using System.Web.Routing;
[assembly: OwinStartup(typeof(AspNetDesignerSample.Startup))]
namespace AspNetDesignerSample
{
    public class Startup
    {
        // resources (reports, themes, images) location
        private static readonly DirectoryInfo ResourcesRootDirectory =
            new DirectoryInfo(String.Format("{0}.\\resources\\",
HttpRuntime.AppDomainAppPath));
        public void Configuration(IAppBuilder app)
        {
            // web designer middleware
            app.UseDesigner(config => config.UseFileStore(ResourcesRootDirectory));
            // static files middlewares
```

```
            var fileSystem = new PhysicalFileSystem(String.Format("{0}.\\wwwroot\\",
HttpRuntime.AppDomainAppPath));
            app.UseDefaultFiles(new DefaultFilesOptions { DefaultFileNames = new[] {
"index.html" }, FileSystem = fileSystem });
            app.UseStaticFiles(new StaticFileOptions { FileSystem = fileSystem });
        }
    }
}
```

14. Create 'resources' folder in your sample project root; you can put your existing reports, themes, and images in this folder.



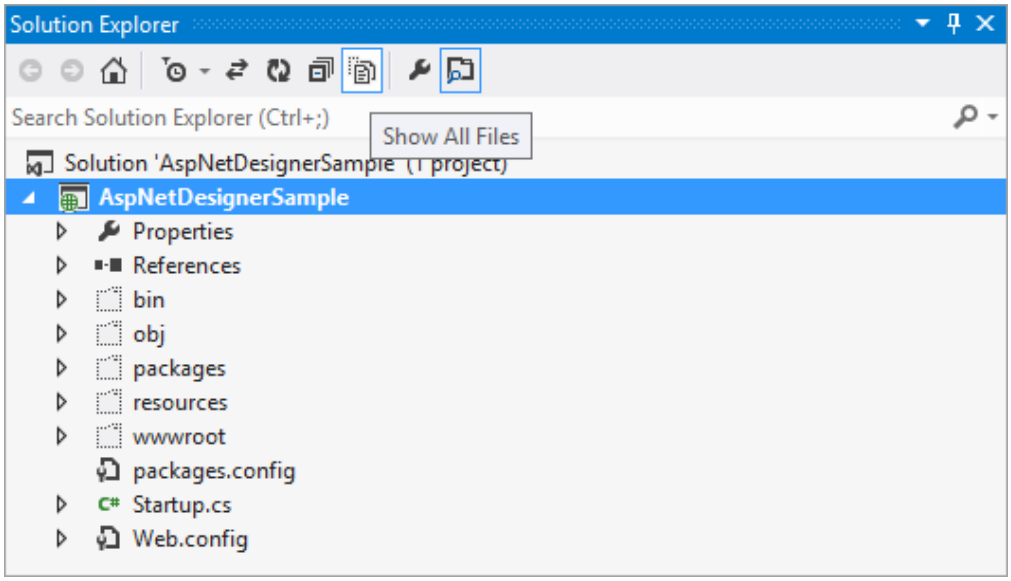
15. Download and install the WebDesigner-related files and folders from **NPM** using the following command in the command line:

```
npm install @grapecity/ar-designer
```

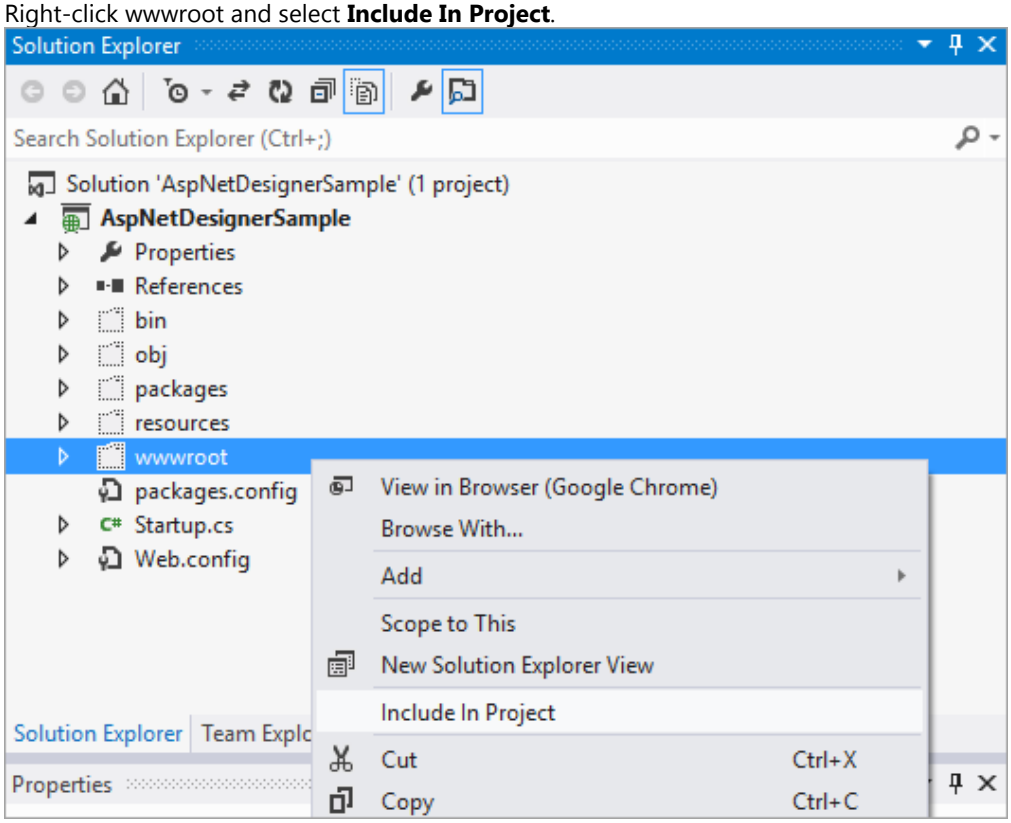
The designer files/folders will be downloaded in your current directory: ..\node_modules\@grapecity\ar-designer\dist

16. Create 'wwwroot' folder in your sample project root. Copy following basic WebDesigner-related files and paste it to your sample project wwwroot subfolder:
    - baseServerApi.js
    - web-designer.css
    - web-designer.js
    - vendor folder

    Optionally you can also copy **file-dialog.css** and **file-dialog.js** if you would like to use our sample dialog component for saving reports.
17. In **Solution Explorer** top bar, check **Show All Files.**

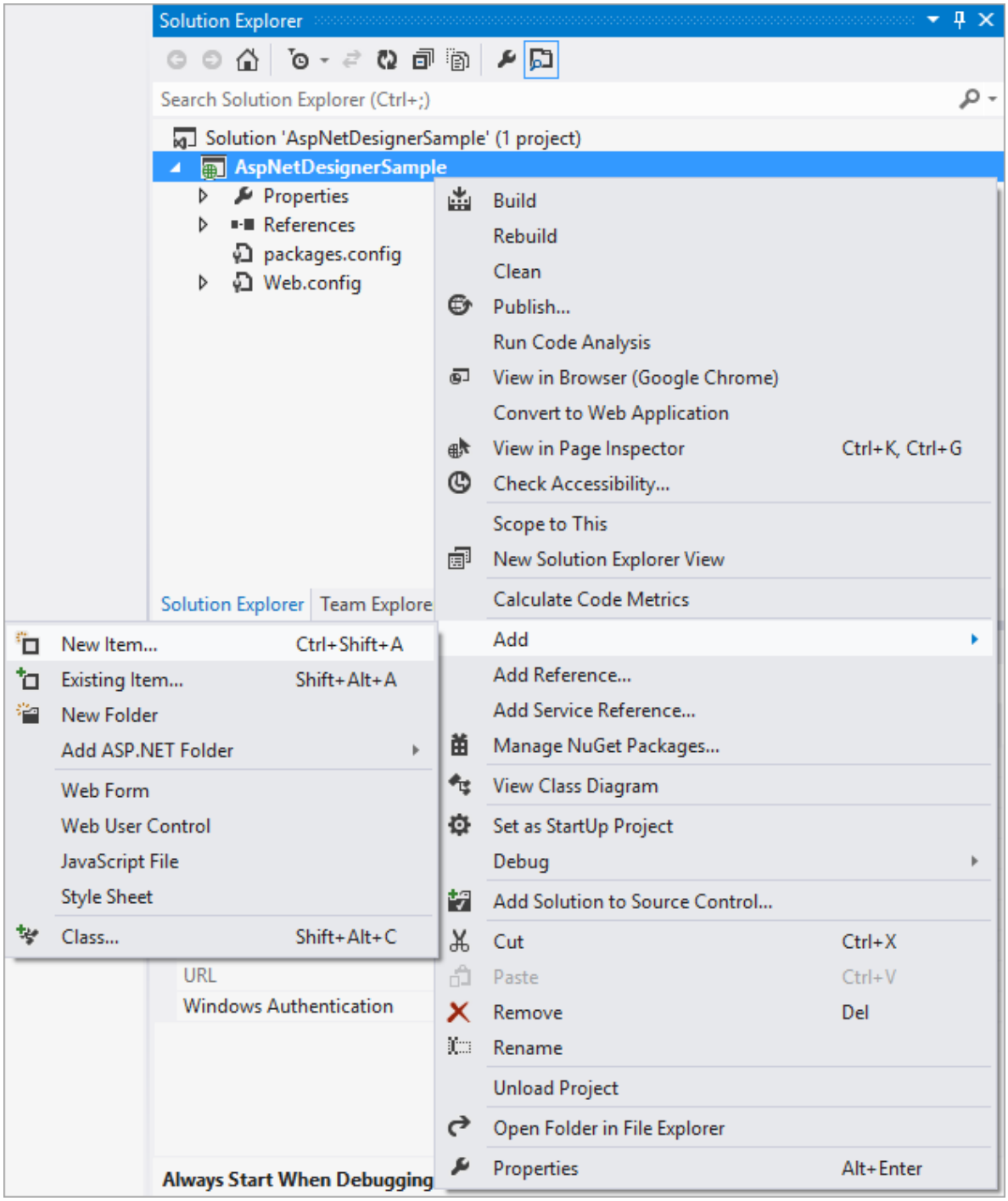


Right-click wwwroot and select **Include In Project**.

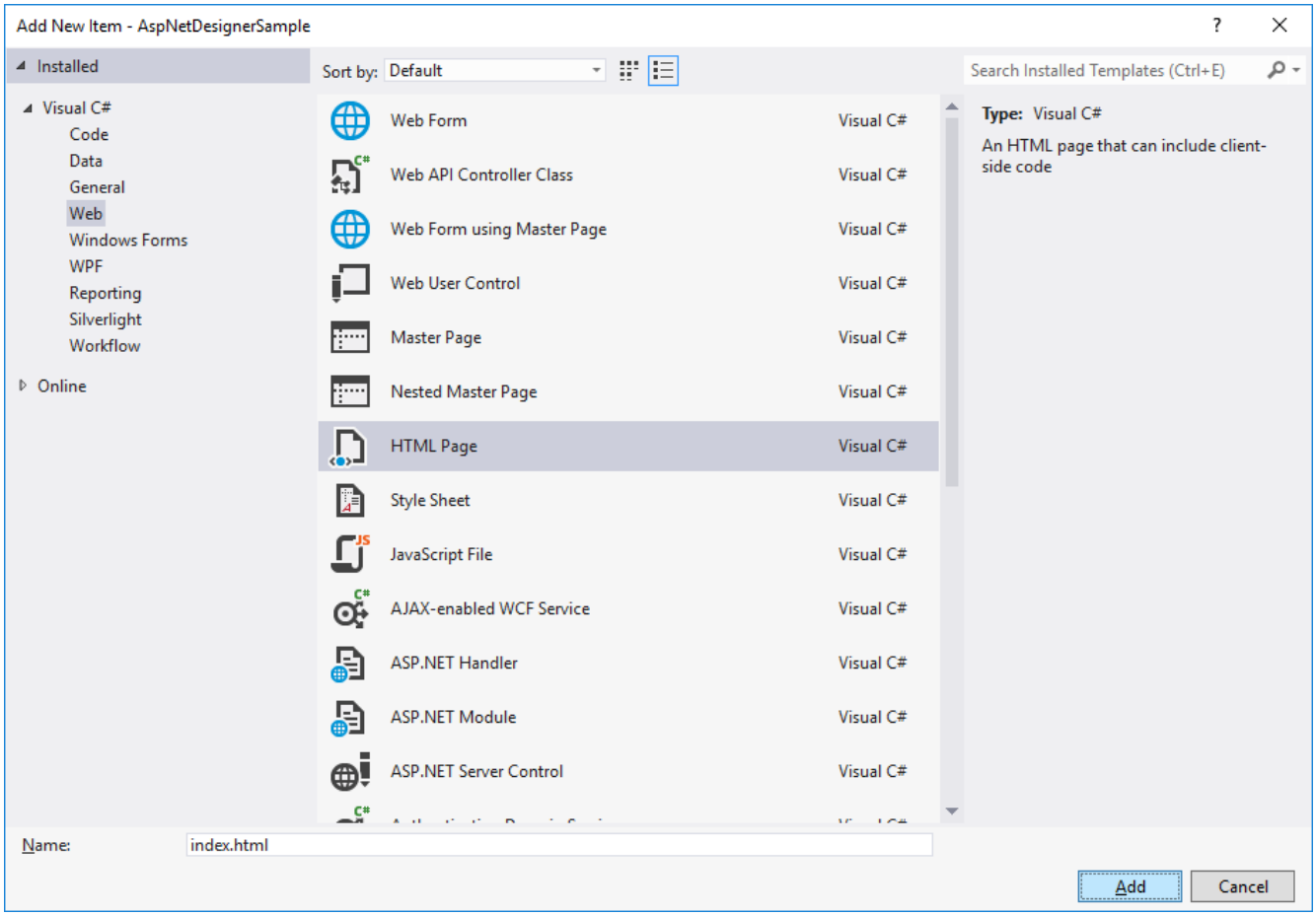


Uncheck **Show All Files**.

18. Right-click wwwroot and select **Add > New Item...**:



In the window that appears, go to **Web > HTML Page**, input **index.html** and click **Add**.



In **Solution Explorer** find newly-added **index.html** and modify its content as follows:

```
<!DOCTYPE html>
<html>
<head>
    <title>Web Designer Sample</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta http-equiv="x-ua-compatible" content="ie=edge">
    <!-- designer-related css -->
    <link rel="stylesheet" href="vendor/css/materialdesignicons.min.css" media="all"
type="text/css" />
    <link rel="stylesheet" href="vendor/css/bootstrap.min.css" />
    <link rel="stylesheet" href="vendor/css/font-awesome.min.css">
    <link rel="stylesheet" href="vendor/css/ionicons.min.css">
    <link rel="stylesheet" href="vendor/css/fonts-googleapis.css" type="text/css">
    <link rel="stylesheet" href="web-designer.css" />
</head>
<body class="theme-blue">
    <!-- designer-related js -->
    <script src="vendor/js/jquery.min.js"></script>
    <script src="vendor/js/bootstrap.min.js"></script>
```

```
    <script src="baseServerApi.js"></script>
    <script src="web-designer.js"></script>
    <!-- designer root div -->
    <div id="designer-id" style="width: 100%; height: 100%;"></div>
    <script>
        // create designer options
        var designerOptions =
GrapeCity.ActiveReports.WebDesigner.createDesignerOptions(baseServerApi);
        // render designer application
        GrapeCity.ActiveReports.WebDesigner.renderApplication('designer-id',
designerOptions);
    </script>
</body>
</html>
```

Build your solution and run it. WebDesigner with a blank RDL report opens in your browser.

19. See steps from **Step 16** of ASP .NET MVC Core application to complete this sample.

Designer Control (Pro Edition)

With the Professional Edition of ActiveReports, you can host the ActiveReports Designer control in your Windows Forms application and provide your end users with report editing capabilities. The control's methods and properties allow you to save and load report layouts, monitor and control the design environment, and customize the look and feel.

In addition to the Designer control, ActiveReports offers a CreateToolStrips method to help you add default toolbars to the designer and add and remove individual tool bars and commands. This gives your designer a finished look and allows you to quickly create a functioning report designer application.

> **Note:** You cannot host the ActiveReports Designer control in the Web application and Web site project types.

Standalone Viewers

ActiveReports provides executable files for the Viewer controls (Windows and WPF) in the startup menu. These executable files function as standalone applications to help view a report quickly.

Use the standalone designer application to create a report layout, save it in .rpx or .rdlx format and then load it in the stand-alone viewer application to view the report.

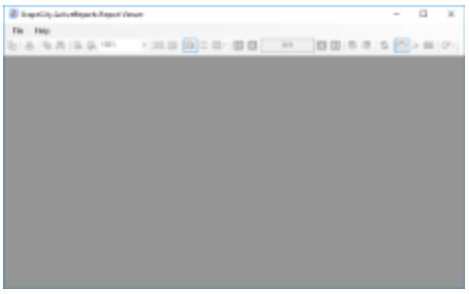
To access the Standalone Viewer application

From the Start Menu, select **ActiveReports 14 Viewer**.

OR

Select the application located under: ..\GrapeCity\ActiveReports 14\Tools.

- GrapeCity.ActiveReports.Viewer.exe
- GrapeCity.ActiveReports.WpfViewer.exe

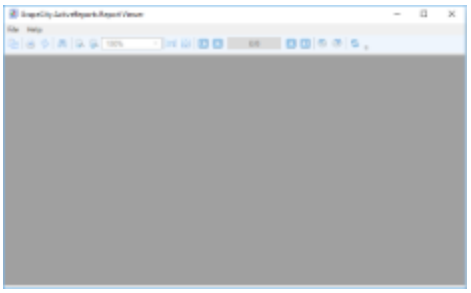


Stand-alone Viewer refers to the GrapeCity.ActiveReports.Viewer.exe bundled with the ActiveReports installer. This is basically a Windows Form application with an ActiveReports Viewer control in it. The default user interface of this application provides an ActiveReports Viewer control along with a menu bar.

You can open a .rdlx or .rpx report in the stand-alone viewer application, by going to the **File** menu > **Open** menu option and selecting a report to load in the viewer. Unlike the Viewer control, no code implementation is required to load the report in the stand-alone application.

Please note that any additional features activated through code like the annotation toolbar, are not available in the stand-alone viewer application. See Windows Forms Viewer for more information on how to implement these features in the Viewer control.

Standalone WPF Viewer



Stand-alone WPF viewer refers to GrapeCity.ActiveReports.WpfViewer.exe bundled with ActiveReports 14 installer. This is basically a WPF application with an ActiveReports WPF Viewer control in it. The default user interface of this application provides an ActiveReports along with a menu bar.

You can open an .rdlx or .rpx report in the stand-alone WPF Viewer application, by going to the File menu > Open menu option and selecting a report to load in the viewer. Unlike the Viewer control, no code implementation is required to load the report in the stand-alone application.

Please note that features like customizing toolbar through code, are not available in the stand-alone WPF Viewer application. See Using the WPF Viewer for more information on how to implement these features in the Viewer control.

Standalone ActiveReports Designer

The stand-alone designer refers to the GrapeCity.ActiveReports.Designer.exe bundled with the ActiveReports installer. This application provides a user interface comprising of the design area at the center along with a toolbox, toolbar, menu, Report Explorer and Properties Window to mimic the Visual Studio look and feel. The executable file for the designer is available in the startup menu.

The stand-alone designer supports all reports - Page, RDL, and Section. By default, a stand-alone designer appears with an RDL layout loaded in the designer. Use the stand-alone designer application to create a report layout, save it in .rpx or

.rdlx format and then load it in the stand-alone viewer application to view the report.

The UI/UX of the Stand-alone designer shipped with ActiveReports 14 in comparison to the previous version, is more modern with improved presentation and accessibility of various menu items. Note that the existing integrated designer in Visual Studio is unchanged, with UI similar to the stand-alone designer of the previous version. See ActiveReports Designer for more information on integrated ActiveReports Designer.

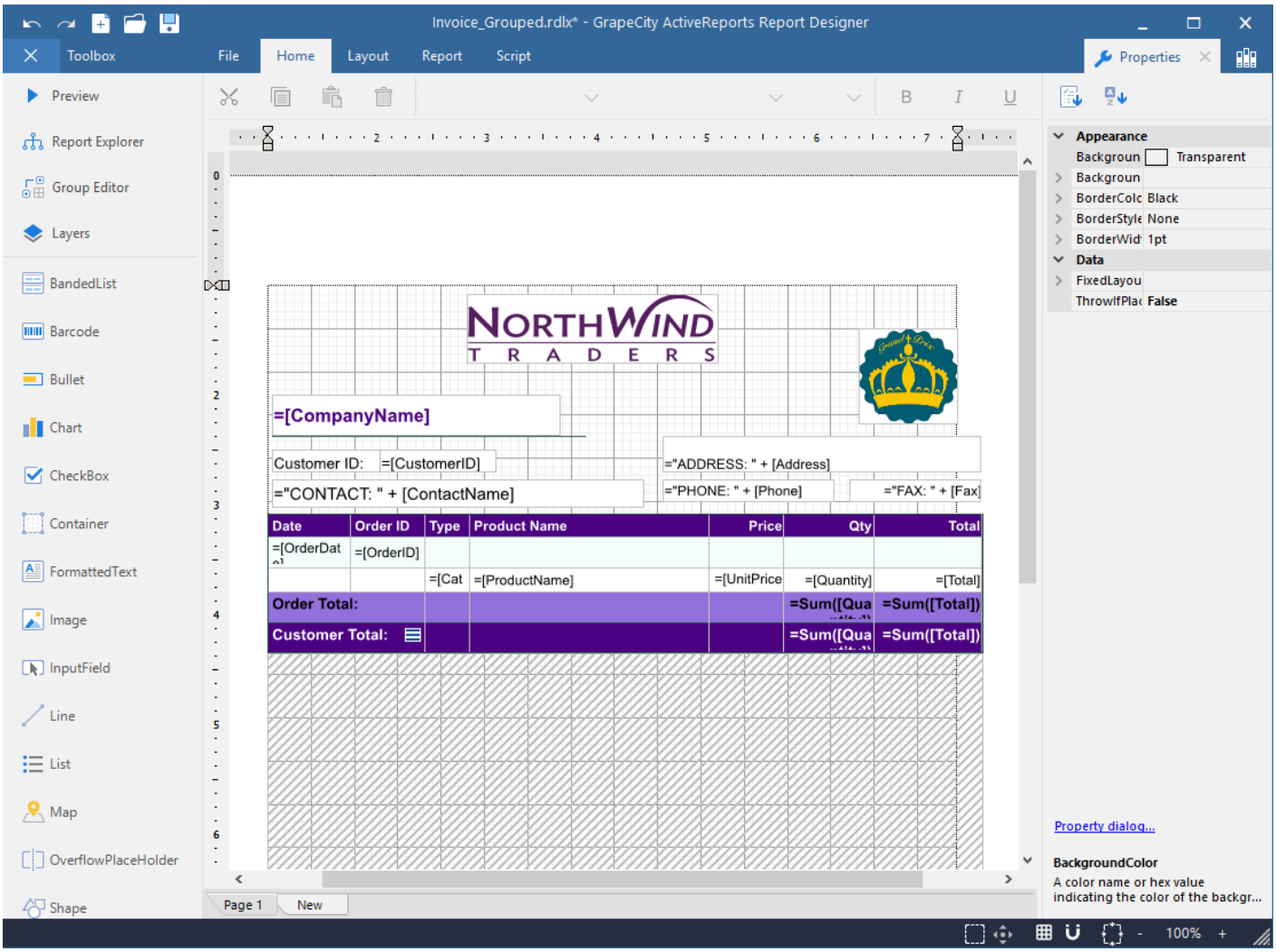
To access the stand-alone designer application:

- From the Start Menu, select **ActiveReports 14 Designer**.

OR

- Select the **GrapeCity.ActiveReports.Designer.exe** application located in the installation folder: ..\GrapeCity\ActiveReports 14\Tools.

Design area

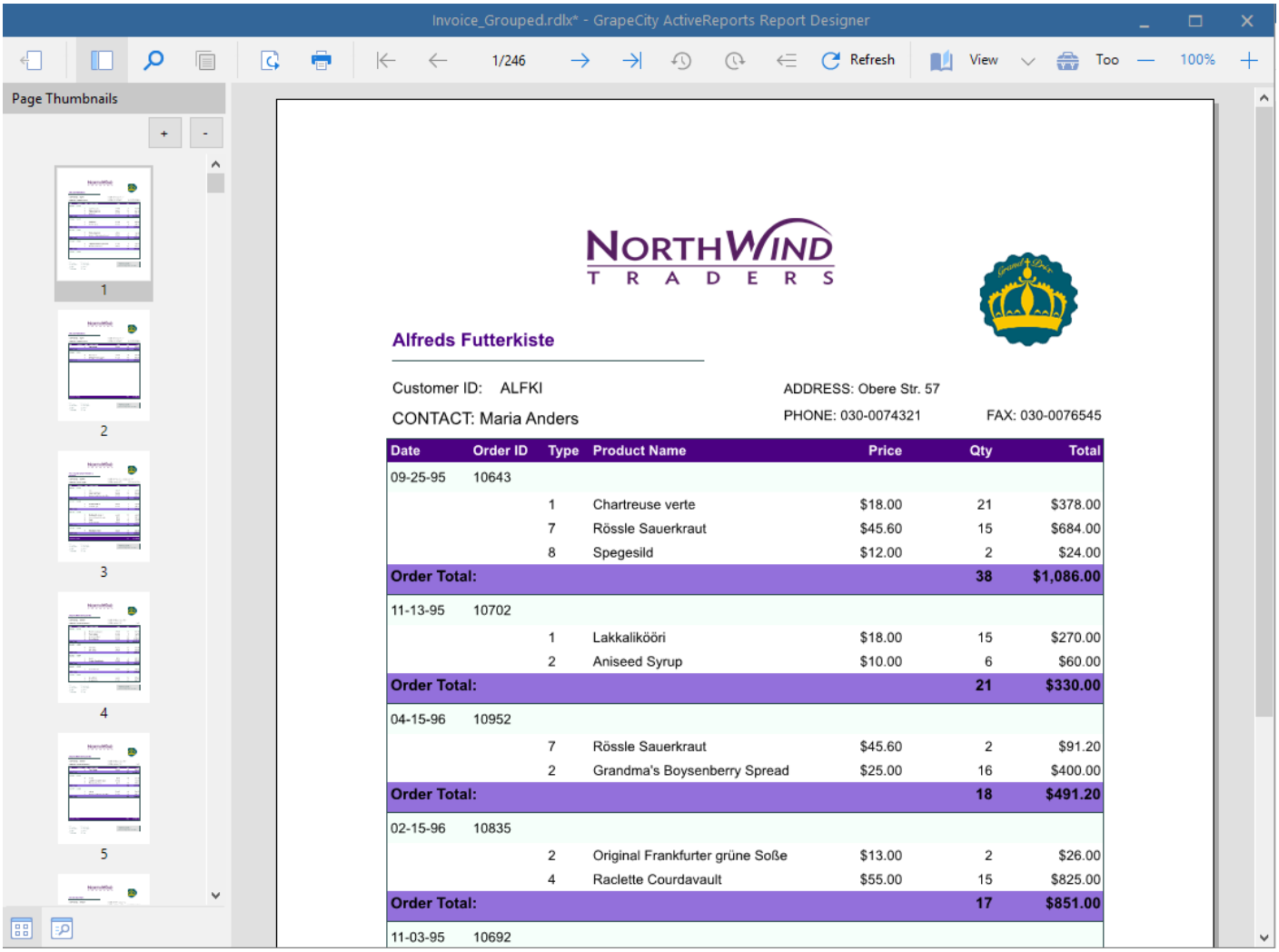


Element	Description

Preview	Shows report preview.
Report Explorer	Provides an overview of the hierarchy of added report items and allows managing data sources, parameters, embedded images, embedded stylesheets, etc.
Group Editor	Shows Column and Row group hierarchies of Tablix members for currently selected Tablix data region. You can also switch between Horizontal and Vertical mode.
Layers	Provides options to add or remove layers and send the Layer back or bring it to the front. You can edit or customize any element only in the layer in which it was added.
Report Controls	Report controls and data regions that can be used while creating a report.
Undo/Redo	Undo or redo actions on designer.
New	Creates a new report - Page report, RDL report, or Section report.
Open	Opens an existing report.
Save/Save As	Saves reports in .rpx or .rdlx format depending on the type of layout you are using.
Ribbon Tabs	• File - Contains options to create, open, save reports or exit the designer. It also contains the version information in the About option. • Home - Consists of report editing options such as cut, copy, paste, and delete. It also provides shortcuts for text formatting such as font, font size, font color, and horizontal and vertical text alignments. • Layout - Contains options to align to grid, size to grid, bring to front, send to back and other sizing and spacing options. • Report - Contains options to define report parameters, embedded images, report properties and add or remove header and footer (RDL report), and change report stylesheets. • Script - Allows you to embed Visual Basic.NET or C# script in reports.
Report Design Surface	Area where you can drag and drop report controls such as the table, chart, textbox and other controls to design your reports.
Properties	Displays the properties of the selected report element. If more than one element is selected, only their common properties are shown.
Library	Allows adding reports to the designer along with its data source, dataset, parameter, etc. The elements of the reports added in the library can be used in creating a report.
Grid Mode	Shows or hides the grid. Grids help in accurate placements of controls.
Zoom Support	Changes the zoom level of the design area by using zoom in (+) and zoom out (-) buttons, or by using shortcuts [Ctrl] + [+] to zoom in and [Ctrl] + [-] to zoom out.
Grid Settings	• Grid Size - Changes the size of the grid. The value should be between 0.025in and 2in. • Snap to Grid: Allows the selected control to snap to the grid at set locations. • Snap to Lines - Allows the selected control to snap to the vertical or horizontal lines relative to the position of other controls. • Dimension Lines - Displays the dimensions of the element when it is being resized.

Actual Size	Restores the actual size of the report.
Pan Mode	Easy report navigation by dragging it up or down.
Select Mode	Selects all the elements in the selected area.

Preview



Element	Description
Back	Back to the designer.
Side Panel	• Page Thumbnails - Displays thumbnails of all the report pages in the side panel. • Search Results - Searches any word or phrase.
Find	Opens Find window to find any word or phrase.
Copy	Copies the selected text on clipboard.
Export	Exports report to various formats like csv, json, jpeg, etc.

Print	Prints report.
First Page	Navigates to the first page of the report.
Previous Page	Navigates to previous page.
Page Number	Navigates to the specific page of the report.
Next Page	Navigates to the next page.
Last Page	Navigates to the last page of the report.
Backward	Navigates to the page you accessed before the current page.
Forward	Navigates to the page from where you accessed the current page.
Refresh	Refreshes the report.
View	• Single Page - Shows one page of a report at a time. • Continuous - Shows all pages of the report one below the other. • Galley - Shows RDL reports by removing automatic page breaks and displaying data in a single scrollable page. • Multipage - Shows multiple pages at one glance in a tabular format.
Tools	• Pan - Easy report navigation by dragging it up or down. • Selection - Select report element(s). • Snapshot - Captures a snapshot and saves it on clipboard.
Zoom Support	Changes the zoom level of the design area by using zoom in (+) and zoom out (-) buttons, or by using shortcuts [Ctrl] + [+] to zoom in and [Ctrl] + [-] to zoom out.

Keyboard Shortcuts

The following shortcuts are available in the designer.

	Keyboard Shortcut	Action
Designing	Ctrl + A	Selects all cells in the Table and Tablix controls. In the List, Body and Container controls, selects all controls in the current container.
	Ctrl + O	Opens the **Open** report dialog.
	Ctrl + S	Opens the **Save** report dialog.
	Ctrl + Z	Undoes the last action.
	Ctrl + Y	Redoes the last action.
	Ctrl + X	Cuts text and controls.
	Ctrl + C	Copies text and controls.

	Ctrl + V	Pastes text and controls.
	Del	Deletes text and controls.
	Left, Right, Up, Down arrow keys	Moves the visible area of the page in the corresponding direction. In the Table, navigates between the cells. When controls inside List and Container controls and in the Body of the report are selected, arrow keys allow moving controls by grid-size. In the Chart Control, arrow keys move data-fields and category-fields.
	Tab	Navigates in the forward direction between the cells in the Table and Tablix controls. When controls inside List and Container controls and in the Body of the report are selected, Tab key switches between controls in the forward direction.
	Shift + Tab	Navigates in the backward direction between the cells in the Table and Tablix controls. When controls inside List and Container controls and in the Body of the report are selected, Shift + Tab switches between controls in the backward direction.
Formatting	Ctrl + B	Makes the text bold.
	Ctrl + I	Makes the text italic.
	Ctrl + U	Underlines the text.
	Ctrl + L	Aligns text to the left.
	Ctrl + E	Aligns text to the center.
	Ctrl + R	Aligns text to the right.
	Ctrl + J	Aligns text justified.
	Ctrl + T	Aligns text to the top.
	Ctrl + M	Aligns text to the middle.
	Ctrl + H	Aligns text to the bottom.
Previewing (F5)	Ctrl + F	Finds a text in the report.
	Ctrl + E	Exports the report.
	Ctrl + P	Prints the report.
	Ctrl + S	Switches view mode to Single page.
	Ctrl + M	Switches view mode to Continuous.
	Ctrl + I	Switches view mode to Multiple page.

Page Report/RDL Report Concepts

There are a number of concepts that apply to page reports and RDL reports.

In this section

Toolbox
: Learn about the report controls and data regions available in the ActiveReports 14 **Page Report** and ActiveReports 14 **RDL Report** group in the Visual Studio toolbox.

Data Sources and Datasets
: Learn about the Data Sources you can access through ActiveReports and fetch data through DataSets along with an overview of the Report DataSource and DataSet dialogs.

Expressions
: Learn about setting expressions in reports and creating expression through the Expression Editor.

Layers
: Learn about using Layers in reports and it's advantages.

Report Appearance
: Learn about using Styles in reports and creating themes to define the appearance of reports for a consistent look.

Report Dialog
: Learn about the various options provided in Report Dialog.

Fixed Page Dialog
: Learn about the various options provided in FixedPage Dialog.

Data Visualizers
: Learn about a number of ways to make your data pop using small graphs in images and background colors.

Custom Resource Locator
: Learn about the ResourceLocator class that allows you to find resources on your machine for use in your RDL reports.

Toolbox

When a Page report or RDL report has focus in Visual Studio, the ActiveReports 14 **Page Report** and ActiveReports 14 **RDL Report** toolbox group offers a number of report controls and data regions that you can use when creating a report. You can drag these from the toolbox and drop them onto your reports. These tools are different than those in the Section Report Toolbox.

> **Note:** Take care in naming report controls, as they are displayed to end users in the advanced search feature of the Viewer.

In this section

Banded List
: The BandedList is a data region with freeform bands in which you can place report controls. With a detail band that repeats data for every row in the dataset, this data region resembles the Section report design surface.

Barcode
: The BarCode report control renders scannable barcodes in any of 39 popular symbologies. You can bind it to data, control the bar width, rotation, quiet zones, caption locations, whether check sum is enabled, and many other properties.

Bullet
: The Bullet report control is an easy-to-read linear gauge that is a good alternative to using a dashboard for data visualization. You can bind it to data and set best, worst, and satisfactory values as well as labels and ranges.

Chart
: The new Chart is a graphic data region similar to Classic Chart, which is based on representing the data using encodings. It is the default chart available in the designer.

Classic Chart
: The Classic Chart is a graphic data region which allows you to display data in a variety of chart styles with 3D effects and colors, and provides many options for customization. You can choose from numerous chart types.

CheckBox (Page Report)
: The CheckBox report control can display Boolean data, or you can set its Checked property. You can also enter static text to display.

Container
: The Container report control is a graphical element that is used as a container for other items. The Container report control has no data associated with it.

Formatted Text
: The FormattedText report control displays data, and allows you to format selected areas of text within the control in different ways. This report control accepts XHTML input, and allows you to set up mail merge.

Image
: The Image report control allows you to specify any image file to display from an external source, a database or an embedded image.

InputField
: The InputField report control provides support for editable fields in an exported PDF report.

Line
: The Line report control, a graphical element that has no data associated with it, visually marks boundaries or highlights specific areas of a report. You can use lines of various weight, color, and style to highlight regions of your reports and to add style and polish.

List
: The List is a freeform data region in which you can place other report controls. It repeats any report control it contains for every record in the dataset.

Map
: The Map data region shows your business data against a geographical background. You can select different types of map, depending on the type of information you want to communicate in your report.

Overflow Place Holder
: The Overflow Placeholder report control is only available with page reports. It is a simple rectangle that you link to a List, BandedList, Tablix, or Table data region to display data that extends beyond one page.

Shape
: The Shape report control, a graphical element that has no data associated with it, allows you to mark visual boundaries or highlight specific areas of a report with rectangles, rounded rectangles, or elliptical shapes. Unlike the Container report control, it cannot contain other controls.

Sparkline
: The Sparkline report control displays a data trend over time in a graph small enough to be used inline, with a height similar to the surrounding text. You can select from line, area, stacked bar, column, and whisker sparkline types.

Subreport
: The Subreport control displays data from a separate report that you specify. You can pass a parameter to the subreport from the main report to filter data displayed in a subreport.

Table
: The Table is a data region that shows data in rows. By default, it has three columns and three rows. Once set at design time, the columns are static, while the rows repeat for each row of data. The default rows are the header, detail, and footer.

TableOfContents
: The TableOfContents (ToC) report control is used to display the document map, an organized hierarchy of the report bookmarks and labels along with their page numbers, in the body of a report. The TableOfContents control allows you to quickly understand and navigate the data inside a report in all viewers that are supported in ActiveReports.

TextBox
: The TextBox displays data, and is the default data region that appears in each cell of a Table or Tablix data region. It is

also the data region that is created automatically when you drag a field from the Data Explorer onto your report. You can use expressions to modify the data that appears in a TextBox.

Tablix

Tablix data region displays data in cells that are arranged in rows and columns. Tablix is mainly a combination of two data regions- table and a matrix.

Banded List

The BandedList data region is a collection of free-form bands. By default, it is composed of three bands: a header, a footer and a detail band. Bound report controls in the detail band repeat for every row of data. The header and footer rows render once at the beginning and end of the BandedList, respectively, and are a good place for titles and grand totals.

Click inside each band to reveal its properties in the Properties window, or click the four-way arrow to select the entire data region and reveal its properties. Properties for this data region include the following.

Band Properties

Property	Description
CanGrow	Change to True to allow the data region to grow vertically to accommodate data.
CanShrink	Change to True to allow the data region to shrink if there is not enough data to fill it.
KeepTogether	Change to True to have ActiveReports attempt to keep all of the data in the band together on one page.
PageBreakAtEnd	Change to True to insert a page break after rendering all of the data in the band.
PageBreakAtStart	Change to True to insert a page break before rendering any of the data in the band.
RepeatOnNewPage	With header and footer bands, repeats the band on every page when the related details span multiple pages.

BandedList Properties

Property	Description
DataSetName	Select the dataset to use in the data region.
DataSetParameters	Specify the parameters for the data set.
KeepTogether	Change to True to have ActiveReports attempt to keep all of the data in the data region together on one page.
NewSection	Change to True to render the data region in a new section.
OverflowName	Select the name of the OverflowPlaceHolder control in which to render data that exceeds the allowed space for the data region on the first page of the report.

You can add group header and group footer bands. Report controls in these bands repeat once for each group instance. You can also nest groups, plus, in RDL reports, you can nest other data regions in any header or footer band. Grouping in the BandedList is similar to grouping in the Table data region. You can provide a grouping expression for each group, and also sort the groups.

> ⚠ **Caution**: You cannot sort the detail data in a BandedList, so any sorting of this type must be done at the query level.

BandedList Dialog

Properties for the BandedList data region are available in the BandedList dialog. To open it, with the BandedList selected on the report, under the Properties Window, click the **Property dialog** link.

The BandedList dialog lets you set properties on the data region with the following pages.

> **Note**: You can click <**Expression...**> in many of these properties to open the Expression Editor where you can create an expression to determine the value.

General

Name: Enter a name for the banded list that is unique within the report. You can only use underscore (_) as a special character in the Name field. Other special characters such as period (.), space ( ), forward slash (/), back slash (\), exclamation (!), and hyphen (-) are not supported.

Tooltip: A textual label for the report item used to include TITLE or ALT attributes in HTML reports.

Dataset name: Select a dataset to associate with the banded list. The combo box is populated with all of the datasets in the report's dataset collection.

Has own page numbering: Select to indicate whether this banded list is in its own section with regards to pagination.

Page Breaks: Select any of the following options to apply to each instance of the banded list.

- Insert a page break before this banded list
- Insert a page break after this banded list
- Fit banded list on a single page if possible

Header and Footer: Select any of the following options.

- Repeat header band on each page
- Repeat footer band on each page

Visibility

By default, the banded list is visible when the report runs, but you can hide it, hide it only when certain conditions are met, or toggle its visibility with another report control.

Initial visibility

- **Visible**: The banded list is visible when the report runs.
- **Hidden**: The banded list is hidden when the report runs.
- **Expression**: Use an expression with a Boolean result to decide whether the BandedList is visible. True for hidden, False for visible.

Visibility can be toggled by another report control: Select this check box to display a toggle image next to another report control. This enables the drop-down box below where you can specify the TextBox control that toggles the visibility of the BandedList. The user can click the toggle item to show or hide this BandedList.

Navigation

Document map label: Enter an expression to use as a label to represent this item in the table of contents (document map).

Bookmark ID: Enter an expression to use as a locator for this BandedList. You will then be able to provide a bookmark link to this item from another report control using a **Jump to bookmark** action.

Groups

Click the plus sign button to add a new group to the BandedList, and delete them using the X button. Once you add one or more groups, you can reorder them using the arrow buttons, and set up information for each group on the following tabs.

General

Name: Enter a name for the group that is unique within the report. This property cannot be set until after a Group on expression is supplied.

Group on: Enter an expression to use for grouping the data.

Document map label: Enter an expression to use as a label to represent this item in the table of contents (document map).

Parent group: For use in recursive hierarchies. Enter an expression to use as the parent group.

Filters

You need to provide three values to add a new filter to the collection: Expression, Operator, and Value.

Expression: Enter the expression to use for evaluating whether data should be included in the group.

Operator: Select from the following operators to decide how to compare the expression to the left with the value to the right.

- **Equal** Only choose data for which the value on the left is equal to the value on the right.
- **Like** Only choose data for which the value on the left is similar to the value on the right.
  For more information on using the **Like** operator, see the MSDN Web site.
- **NotEqual** Only choose data for which the value on the left is not equal to the value on the right.
- **GreaterThan** Only choose data for which the value on the left is greater than the value on the right.
- **GreaterThanOrEqual** Only choose data for which the value on the left is greater than or equal to the value on the right.
- **LessThan** Only choose data for which the value on the left is less than the value on the right.
- **LessThanOrEqual** Only choose data for which the value on the left is less than or equal to the value on the right.
- **TopN** Only choose items from the value on the left which are the top number specified in the value on the right.
- **BottomN** Only choose items from the value on the left which are the bottom number specified in the value on the right.
- **TopPercent** Only choose items from the value on the left which are the top percent specified in the value on the right.
- **BottomPercent** Only choose items from the value on the left which are the bottom percent specified in the value on the right.
- **In** Only choose items from the value on the left which are in the array of values specified on the right.
  Selecting this operator enables the Values list at the bottom.
- **Between** Only choose items from the value on the left which fall between the pair of values you specify on the right. Selecting this operator enables two Value boxes instead of one.

Value: Enter a value to compare with the expression on the left based on the selected operator. For multiple values used with the **Between** operator, the lower two value boxes are enabled.

Values: When you choose the **In** operator, you can enter as many values as you need in this list.

Sorting

Click the plus sign button to enter new sort expressions, and remove them using the X button.

Expression: Enter an expression by which to sort the data in the group.

Direction: Select Ascending or Descending.

Visibility

By default, the group is visible when the report runs, but you can hide a group, hide it when certain conditions are met, or toggle its visibility with another report control.

Initial visibility

- **Visible**: The group is visible when the report runs.
- **Hidden**: The group is hidden when the report runs.
- **Expression**: Use an expression with a Boolean result to decide whether the group is visible. True for hidden, False for visible.

Visibility can be toggled by another report control: Select this check box to display a toggle image next to another report item. The user can click the toggle item to show or hide this band group. This enables the drop-down list where you can select the report control that users can click to show or hide this group.

Data Output

Element name: Enter a name to be used in the XML output for this group.

Collection: Enter a name to be used in the XML output for the collection of all instances of this group.

Output: Choose **Yes** or **No** to decide whether to include this group in the XML output.

Layout

Page break at start: Inserts a page break before the group.

Page break at end: Inserts a page break after the group.

Include group header: Adds a group header band (selected by default).

Include group footer: Adds a group footer band (selected by default).

Repeat group header: Repeats the group header band on each page.

Repeat group footer: Repeats the group footer band on each page.

Has own page numbering: Used in conjunction with the "Page Number in Section" and "Total Pages in Section" properties, tells the report that the group constitutes a new page numbering section.

Filters

You need to provide three values to add a new filter to the collection: Expression, Operator, and Value.

Expression: Enter the expression to use for evaluating whether data should be included in the group.

Operator: Select from the following operators to decide how to compare the expression to the left with the value to the right.

- **Equal** Only choose data for which the value on the left is equal to the value on the right.
- **Like** Only choose data for which the value on the left is similar to the value on the right.
  For more information on using the **Like** operator, see the MSDN Web site.

- **NotEqual** Only choose data for which the value on the left is not equal to the value on the right.
- **GreaterThan** Only choose data for which the value on the left is greater than the value on the right.
- **GreaterThanOrEqual** Only choose data for which the value on the left is greater than or equal to the value on the right.
- **LessThan** Only choose data for which the value on the left is less than the value on the right.
- **LessThanOrEqual** Only choose data for which the value on the left is less than or equal to the value on the right.
- **TopN** Only choose items from the value on the left which are the top number specified in the value on the right.
- **BottomN** Only choose items from the value on the left which are the bottom number specified in the value on the right.
- **TopPercent** Only choose items from the value on the left which are the top percent specified in the value on the right.
- **BottomPercent** Only choose items from the value on the left which are the bottom percent specified in the value on the right.
- **In** Only choose items from the value on the left which are in the array of values specified on the right. Selecting this operator enables the Values list at the bottom.
- **Between** Only choose items from the value on the left which fall between the pair of values you specify on the right. Selecting this operator enables two Value boxes instead of one.

Value: Enter a value to compare with the expression on the left based on the selected operator. For multiple values used with the **Between** operator, the lower two value boxes are enabled.

Values: When you choose the **In** operator, you can enter as many values as you need in this list.

Data Output

The Data Output page of the BandedList dialog allows you to control the following properties when you export to XML.

- **Element name**: Enter a name to be used in the XML output for this BandedList.
- **Output**: Choose **Auto**, **Yes**, or **No** to decide whether to include this BandedList in the XML output. Choosing **Auto** exports the contents of the BandedList.

Barcode

The **Barcode** report control offers various barcode styles to choose from. This saves you the time and expense of finding and integrating a separate component. As with other data-bound report controls, you can use an expression to bind the value of a field to the Barcode **Value** property.

Apart from the barcode style, you can manage the alignment, direction, color, background color, bar width, caption position, font, text, and check whether checksum is enabled in the Properties Window. There are more properties available with the Code49, PDF417, and QRCode barcode styles. Click the Barcode to reveal its properties in the Properties window. All of the properties specific to this report control are also available in the Barcode dialog.

> **Note**: This barcode is ported from the Section report Barcode control, so if you create reports programmatically, the Page report/RDL report barcode is treated as a CustomReportItem.

Barcode Dialog

Properties for the Barcode are available in the Barcode dialog. To open it, with the Barcode selected in the report, under the Properties Window, click the **Property dialog** link.

The Barcode dialog lets you set properties on the report control with the following pages.

> **Note**: You can select the **<Expression...>** option in many of these properties to open the Expression Editor where you can create an expression to determine the value. You can also access the **Expression Editor** from the context menu of the Barcode control.

General

Name: Enter a name for the barcode that is unique within the report. You can only use underscore (_) as a special character in the Name field. Other special characters such as period (.), space ( ), forward slash (/), back slash (\), exclamation (!), and hyphen (-) are not supported.

Tooltip: A textual label for the report item used to include TITLE or ALT attributes in HTML reports.

Value: Enter an expression or a static label, or choose a field expression from the drop-down list. You can access the expression editor by selecting **<Expression...>** in the list. The value of this expression or text is used to render the barcode in the report.

Invalid Barcode Text: Enter a message to display if the barcode contains invalid values (content, character, length).

Caption

Location: Select whether to display the caption above or below the barcode, or select **None** to display the barcode without a caption.

Text Alignment: Select the horizontal alignment of the caption. The default value of **General** centers the caption.

Barcode Settings

Symbology: Enter the type of barcode to use. ActiveReports supports all of the most popular symbologies:

Table of all included symbologies

> **Notes**: The RSS and QRCode styles have fixed height-to-width ratios. When you resize the width, the height is automatically calculated.
>
> When you choose a style that offers supplemental options, the additional options appear below.

Symbology Name	Example	Description
Ansi39		ANSI 3 of 9 (Code 39) uses upper case, numbers, - , * $ / + %. This is the default barcode style.
Ansi39x		ANSI Extended 3 of 9 (Extended Code 39) uses the complete ASCII character set.
BC412		Data BC412 uses 35 characters, 0 - 9 and A - Z. It is designed for semiconductor wafer identification.
Codabar		Codabar uses A B C D + - : . / $ and numbers.

Code_11		Encodes the numbers 0 through 9, the hyphen (-), and start/stop characters. It is primarily used in labeling telecommunications equipment.
Code_128_A		Code 128 A uses control characters, numbers, punctuation, and upper case.
Code_128_B		Code 128 B uses punctuation, numbers, upper case and lower case.
Code_128_C		Code 128 C uses only numbers.
Code_128auto		Code 128 Auto uses the complete ASCII character set. Automatically selects between Code 128 A, B and C to give the smallest barcode.
Code_2_of_5		Code 2 of 5 uses only numbers.
Code_93		Code 93 uses uppercase, % $ * / , + -, and numbers.
Code25intlv		Interleaved 2 of 5 uses only numbers.
Code39		Code 39 uses numbers, % * $ /. , - +, and upper case.
Code39x		Extended Code 39 uses the complete ASCII character set.
Code49		**Code 49** is a 2D high-density stacked barcode containing two to eight rows of eight characters each. Each row has a start code and a stop code. Encodes the complete ASCII character set.

Code93x		Extended Code 93 uses the complete ASCII character set.
DataMatrix		**Data Matrix** is a high density, two-dimensional barcode with square modules arranged in a square or rectangular matrix pattern.
EAN_13		**EAN-13** uses only numbers (12 numbers and a check digit). It takes only 12 numbers as a string to calculate a check digit (CheckSum) and add it to the thirteenth position. The check digit is an additional digit used to verify that a bar code has been scanned correctly. The check digit is added automatically when the CheckSum property is set to True.
EAN_13 with the add-on code		**EAN-13** may include the add-on code to the right of the main code. The add-on code may include up to 5 supplemental characters.
EAN_8		**EAN-8** uses only numbers (7 numbers and a check digit).
EAN128FNC1		**EAN-128FNC1** is an alphanumeric one-dimensional representation of Application Identifier (AI) data for marking containers in the shipping industry. This barcode is now obsolete. You should use UCC/EAN-128 instead which provides similar functionality with better performance.
GS1QRCode		GS1QRCode is a subset of the QR Code. The GS1 QR Code is a 2D symbol that denotes the Extended Packaging URL for a trade item. It is processed to obtain one URL address associated with the trade item identified by the Global Trade Item Number (GTIN). GS1 QR Code requires the mandatory association of the GTIN and Extended Packaging URL. GS1 QR Code allows to encode GS1 System

		Application Identifiers (AI) into QR Code 2D barcodes. **Limitation:** Kanji, CN, JP and Korean characters.
HIBCCode128		HIBCCode128 barcode uses the Code128 symbology. It encodes 'Primary Data' and 'Secondary Data' using slash (/) as delimiter. It is used in the health care products industry for identification purpose.
HIBCCode39		HIBCCode39 barcode uses the Code39 symbology, with the Code39OptionalCheckDigit property set to True. It encodes Primary Data and Secondary Data using slash (/) as delimiter. It is used in the health care products industry for identification purpose.
IATA_2_of_5		IATA_2_of_5 is a variant of Code_2_of_5 and uses only numbers with a check digit.
IntelligentMail		Intelligent Mail, formerly known as the 4-State Customer Barcode, is a 65-bar code used for domestic mail in the U.S.
IntelligentMailPackage		IntelligentMailPackage is more efficient in terms of processing and tracking mails than Intelligent Mail barcode.
ISBN		International Standard Book Number barcode is a special form of the EAN-13 code and is used as a unique 9-digit commercial book identifier.
ISMN		Internationally Standard Music Number barcode is a special form of the EAN-13 code. It is used for marking printed musical publications.
ISSN		International Standard Serial Number barcode is a special form of the EAN-13 code. It is used to identify serial publications, publications that are issued in numerical order, such as the volumes of a magazine.

ITF14		Interleaved Two of Five code is used to mark cartons that contain goods with an EAN-13 code. One digit is added in front of the EAN-13 code to mark the packing variant.
JapanesePostal		This is the barcode used by the Japanese Postal system. Encodes alpha and numeric characters consisting of 18 digits including a 7-digit postal code number, optionally followed by block and house number information. The data to be encoded can include hyphens.
Matrix_2_of_5		Matrix 2 of 5 is a higher density barcode consisting of 3 black bars and 2 white bars.
MaxiCode		MaxiCode is special polar barcode that uses 256 characters. It is used to encode a specific amount of data.
MicroPDF417		**MicroPDF417** is two-dimensional (2D), multi-row symbology, derived from PDF417. Micro-PDF417 is designed for applications that need to encode data in a two-dimensional (2D) symbol (up to 150 bytes, 250 alphanumeric characters, or 366 numeric digits) with the minimal symbol size. MicroPDF417 allows you to insert an FNC1 character as a field separator for variable length Application Identifiers (AIs). To insert FNC1 character, set “\n” for C#, or “vbLf” for VB to Text property at run time.
MicroQRCode		**MicroQRCode** is a two-dimensional (2D) barcode that is designed for applications that use a small amount of data. It can handle numeric and alphanumeric data as well as Japanese kanji and kana characters. This symbology can encode up to 35 numeric characters.

MSI		MSI Code uses only numbers.
Pdf417		Pdf417 is a popular high-density 2-dimensional symbology that encodes up to 1108 bytes of information. This barcode consists of a stacked set of smaller barcodes. This symbology can encode up to 35 alphanumeric characters or 2,710 numeric characters.
Pharmacode		Pharmacode represents only numeric data from 3 to 131070. It is a barcode standard used in the pharmaceutical industry for packaging. It is designed to be readable despite printing errors.
Plessey		Plessey uses hexadecimal digits to encode. It is a one-dimensional barcode used mainly in libraries.
PostNet		PostNet uses only numbers with a check digit.
PZN		Pharmaceutical Central/General Number uses the same encoding algorithm as Code 39 but can carry only digits – 0123456789. The number of digits supported for encoding are 6 or 7. The letters 'PZN' and checksum digit are automatically added. It is mainly used to identify medicine and health-care products in Germany and other German-speaking countries.
QRCode		**QRCode** is a 2D symbology that is capable of handling numeric, alphanumeric and byte data as well as Japanese kanji and kana characters. This symbology can encode up to 7,366 characters.
RM4SCC		Royal Mail (RM4SCC) uses only letters and numbers (with a check digit). This is the barcode used by the Royal Mail in the United Kingdom.

RSS14		RSS14 is a 14-digit Reduced Space Symbology that uses EAN.UCC item identification for point-of-sale omnidirectional scanning. The RSS family of barcodes is also known as **GS1 DataBar**.
RSS14Stacked		RSS14Stacked uses the EAN.UCC information with Indicator digits as in the RSS14Truncated, but stacked in two rows for a smaller width. RSS14Stacked allows you to set Composite Options, where you can select the type of the barcode in the **Type** drop-down list and the value of the composite barcode in the **Value** field.
RSS14Stacked CCA		RSS14Stacked with Composite Component - Version A.
RSS14StackedOmnidirectional		RSS14StackedOmnidirectional uses the EAN.UCC information with omnidirectional scanning as in the RSS14, but stacked in two rows for a smaller width.
RSS14Truncated		RSS14Truncated uses the EAN.UCC information as in the RSS14, but also includes Indicator digits of zero or one for use on small items not scanned at the point of sale.
RSSExpanded		RSSExpanded uses the EAN.UCC information as in the RSS14, but also adds AI elements such as weight and best-before dates. RSSExpanded allows you to insert an FNC1 character as a field separator for variable length Application Identifiers (AIs). To insert FNC1 character, set “\n” for C#, or “vbLf” for VB to Text property at run time.

RSSExpandedStacked		**RssExpandedStacked** uses the EAN.UCC information with AI elements as in the RSSExpanded, but stacked in two rows for a smaller width. RSSExpandedStacked allows you to insert an FNC1 character as a field separator for variable length Application Identifiers (AIs). To insert FNC1 character, set “\n” for C#, or “vbLf” for VB to Text property at run time.
RSSLimited		RSS Limited uses the EAN.UCC information as in the RSS14, but also includes Indicator digits of zero or one for use on small items not scanned at the point of sale. RSSLimited allows you to set Composite Options, where you can select the type of the barcode in the **Type** drop-down list and the value of the composite barcode in the **Value** field.
RSSLimited CCA		RSS Limited with Composite Component - Version A.
SSCC_18		SSCC_18 is an 18-digit Serial Shipping Container Code. It is used to identify individual shipping containers for tracking purposes.
Telepen		Telepen has 2 different modes - alphanumeric-only and numeric-only. Both modes require a start character, a check digit, and a stop character. It is mainly used in manufacturing industries.
UCCEAN128		UCC/EAN-128 complies to GS1-128 standards. GS1-128 uses a series of Application Identifiers to encode data. This barcode uses the complete ASCII character set. It also uses FNC1 character as the first character position. Using AI's, it encodes best before dates, batch numbers, weights, and more such attributes. It is also used in HIBC applications.
UPC_A		**UPC-A** uses only numbers (11 numbers and a check digit).

UPC_A with the add-on code		**UPC-A** may include the add-on code to the right of the main code. The add-on code may include up to 5 supplemental characters.
UPC_E0		**UPC-E0** uses only numbers. Used for zero-compression UPC symbols. For the Caption property, you may enter either a six-digit UPC-E code or a complete 11-digit (includes code type, which must be zero) UPC-A code. If an 11-digit code is entered, the Barcode control will convert it to a six-digit UPC-E code, if possible. If it is not possible to convert from the 11-digit code to the six-digit code, nothing is displayed.
UPC_E0 with the add-on code		**UPC-E0** may include the add-on code to the right of the main code. The add-on code may include up to 5 supplemental characters.
UPC_E1		**UPC-E1** uses only numbers. Used typically for shelf labeling in the retail environment. The length of the input string for U.P.C. E1 is six numeric characters.
UPC_E1 with the add-on code		**UPC-E1** may include the add-on code to the right of the main code. The add-on code may include up to 5 supplemental characters.

When you choose a symbology which offers supplemental options, the additional options appear below the Symbology drop-down box.

Bar Height: Enter a value in inches (for example, .25in) for the height of the barcode.

Narrow Bar Width (also known as X dimension): This is a value defining the width of the narrowest part of the barcode. Before using an extremely small value for this width, ensure that the scanner can read it. This value is specified in Length units (for example, '10cm', '4mm', '1in').

Tip: For accurate scanning, the quiet zone should be ten times the Narrow Bar Width value.

Narrow Width Bar Ratio (also known as N dimension): Enter a value to define the multiple of the ratio between the narrow and wide bars in symbologies that contain bars in only two widths. For example, if it is a 3 to 1 ratio, this value is 3. Commonly used values are 2, 2.5, 2.75, and 3.

Quiet Zone

A quiet zone is an area of blank space on either side of a barcode that tells the scanner where the symbology starts and stops.

Left: Enter a size in inches of blank space to leave to the left of the barcode.

Right: Enter a size in inches of blank space to leave to the right of the barcode.

Top: Enter a size in inches of blank space to leave at the top of the barcode.

Bottom: Enter a size in inches of blank space to leave at the bottom of the barcode.

Note: The units of measure listed for all of these properties are the default units of measure used if you do not specify. You may also specify **cm**, **mm**, **in**, **pt**, or **pc**.

Checksum

A checksum provides greater accuracy for many barcode symbologies.

Compute Checksum: Select whether to automatically calculate a checksum for the barcode.

Note: If the symbology you choose requires a checksum, setting this value to **False** has no effect.

Code49 Options

Code49 Options are available for the Code49 barcode style.

Use Grouping: Indicates whether to use grouping for the Code49 barcode. The possible values are **True** or **False**.

Group Number: Enter a number between 0 and 8 for the barcode grouping.

DataMatrix Options

DataMatrix Options are available for the DataMatrix barcode style.

EccMode: Select the Ecc mode from the drop-down list. The possible values are **ECC000**, **ECC050**, **ECC080**, **ECC100**, **ECC140** or **ECC200**.

Ecc200 Symbol Size: Select the size of the ECC200 symbol from the drop-down list. The default value is **SquareAuto**.

Ecc200 Encoding Mode: Select the encoding mode for ECC200 from the drop-down list. The possible values are **Auto**, **ASCII**, **C40**, **Text**, **X12**, **EDIFACT** or **Base256**.

Ecc000_140 Symbol Size: Select the size of the ECC000_140 barcode symbol from the drop-down list.

Structured Append: Select whether the barcode symbol is part of the structured append symbols. The possible values are **True** or **False**.

Structure Number: Enter the structure number of the barcode symbol within the structured append symbols.

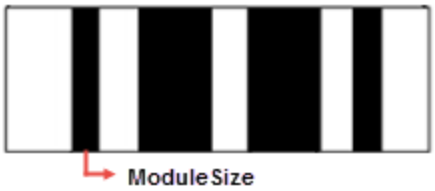
File Identifier: Enter the file identifier of a related group of the structured append symbols. If you set the value to 0, the file identifier symbols are calculated automatically.

EAN128FNC1 Options

EAN128FNC1 Options are available for the EAN128FNC1 barcode style.

DPI: Specify the resolution of the printer as dots per inch to create an optimized barcode image with the specified Dpi value.

Module Size: Enter the horizontal size of the barcode module. Module size is the width of the narrowest bar.



Bar Adjust: Enter the adjustment size by dot units, which affects the size of the module and not the entire barcode.

GS1Composite Options

GS1Composite Options are available for the RSS14Stacked and RSSLimited barcode styles.

Type: Select the type of the composite barcode from the drop-down list. The possible values are **None** or **CCA**. CCA (Composite Component - Version A) is the smallest variant of the 2-dimensional composite component.

Value: Enter the expression to set the value of the composite barcode.

MaxiCode Options

MaxiCode option to select mode is available for MaxiCode barcode.

Mode: Select the mode of the MaxiCode barcode. The available values are Mode2 to Mode6.

MicroPDF417 Options

MicroPDF417 Options are available for the MicroPDF417 barcode style.

Compaction Mode: Select the type of the compaction mode from the drop-down list. The possible values are **Auto**, **TextCompactionMode**, **NumericCompactionMode,** or **ByteCompactionMode**.

Version: Select the version from the drop-down box to set the symbol size.

Segment Index: The segment index of the structured append symbol. The valid value is from 0 to 99998, and less than the value in Segment Count.

Segment Count: The segment count of the structured append symbol. The valid value is from 0 to 99999.

File ID: The file id of the structured append symbol. The valid value is from 0 to 899.

PDF417 Options

PDF417 Options are available for the Pdf417 barcode style.

Columns: Enter column numbers for the barcode.

Rows: Enter row numbers for the barcode.

Error Correction Level: Enter the error correction level for the barcode.

PDF 417 Barcode Type: Select the PDF417 barcode type form the drop-down list. The possible values are **Normal** or **Simple**.

MicroQRCode Options

MicroQRCode Options are available for the MicroQRCode barcode style.

Error Level: Select the error correction level for the barcode from the drop-down list. Valid values are **M**, **L**, or **Q**. The available Error Level values change depending on the version you select.

Version: Enter the version of the MicroQRCode barcode style. Valid values are **M1**, **M2**, **M3**, or **M4.** The maximum amount of data can be stored in version M4.

Mask: Select the pattern for the barcode masking from the drop-down list. Valid values are **Mask00**, **Mask01**, **Mask10**, or **Mask11**.

Encoding: Select the barcode encoding from the drop-down list.

QRCode Options

QRCode Options are available for the QRCode barcode style.

Model: Select the model for the QRCode barcode style from the drop-down list. The possible values are **Model1** or **Model2**. For **GS1QRCode**, Model1 is not supported.

Error Level: Select the error correction level for the barcode from the drop-down list. The possible values are **M**, **L**, **H** or **Q**.

Version: Enter the version of the QRCode barcode style.

- For Model1, valid values for Version are -1 or between 1 to 14.
- For Model2, valid values for Version are -1 or between 1 to 40.

Mask: Select the pattern for the barcode masking from the drop-down list.

Connection: Select whether to use the connection for the barcode. The possible values are **True** or **False**. This property is not applicable to GS1QRCode barcode.

ConnectionNumber: Enter the connection number for the barcode. This property is not applicable to GS1QRCode barcode.

Encoding: Select the barcode encoding from the drop-down list.

RssExpandedStacked Options

RssExpandedStacked Options are available for the RSSExpandedStacked barcode style.

Row Count: Enter the number of the barcode stacked rows.

Supplementary Options

Supplementary Options are available for UPC_A, UPC_E0, UPC_E1, EAN_13, and EAN_8 barcode styles.

Supplement Value: Enter the expression to set the value of the barcode supplement.

Caption Location: Select the location for the supplement caption from the drop-down list. The possible values are **None**, **Above** or **Below**.

Supplement Bar Height: Enter the bar height for the barcode supplement.

Supplement Spacing: Enter the spacing between the main and supplement barcodes.

Appearance

Font

Family: Select a font family name or a theme font.

Size: Choose the size in points for the font or use a theme.

Style: Choose **Normal** or **Italic** or select a theme.

Weight: Choose from **Lighter**, **Thin**, **ExtraLight**, **Light**, **Normal**, **Medium**, **SemiBold**, **Bold**, **ExtraBold**, **Heavy**, or **Bolder**.

Color: Choose a color to use for the text.

Decoration: Choose from **None**, **Underline**, **Overline**, or **LineThrough**.

Border

Style: Select a style for the border.

Width: Enter a value in points to set the width of the border.

Color: Select a color to use for the border, or select the **<Expression...>** option to open the Expression Editor and create an expression that evaluates to a .NET color.

Background

Color: Select a color to use for the background, or select the **<Expression...>** option to open the Expression Editor and create an expression that evaluates to a .NET color.

Format

Format code: Select one of the common numeric formats provided or use a custom .NET formatting code to format dates or numbers. For more information, see MSDN's Formatting Types topic.

Amount of space to leave around report control

Top margin: Set the top padding in points.

Left margin: Set the left padding in points.

Right margin: Set the right padding in points.

Bottom margin: Set the bottom padding in points.

Rotation: Choose **None, Rotate90Degrees**, **Rotate180Degrees**, or **Rotate270Degrees.**

Visibility

Initial visibility

- **Visible**: The barcode is visible when the report runs.
- **Hidden**: The barcode is hidden when the report runs.
- **Expression**: Use an expression with a Boolean result to decide whether the barcode is visible.

Visibility can be toggled by another report control: Select this check box to display a toggle image next to another report control. This enables the drop-down box where you can select the TextBox control that users can click to show or hide this barcode in the viewer.

Navigation

Document map label: Enter an expression to use as a label to represent this item in the table of contents (document map).

Bookmark ID: Enter an expression to use as a locator for this barcode. You will then be able to provide a bookmark link to this item from another report control using a **Jump to bookmark** action.

Data Output

Element Name: Enter a name to be used in the XML output for this barcode.

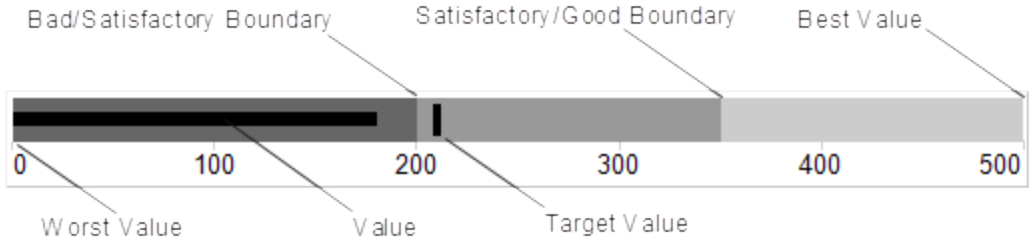
Output: Choose **Auto**, **Yes**, or **No** to decide whether to include this barcode in the XML output. **Auto** exports the contents of the barcode only when the value is not a constant.

Render as: Choose **Auto**, **Element**, or **Attribute** to decide whether to render barcodes as Attributes or Elements in the exported XML file. **Auto** uses the report's setting for this property.

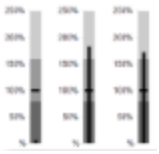
Bullet

The Bullet report control is an easy-to-read linear gauge that is a good alternative to using a dashboard for data visualization.

A bullet graph has a pointer that shows a key measure. With this control, you can take a single value, the year-to-date revenue for example, and compare it to a target value that you define in the control's properties. You can also define the beginning of the graph as the worst value and the end of the graph as the best value. To make the data visualization even more intuitive, you can define a qualitative range (bad, satisfactory and good) for segments on the bullet graph and immediately see the position of the key measure within the bullet graph range.



You can combine multiple Bullets into a data region, a table for example, to show single values side by side. You can orient Bullets horizontally or vertically, and put them together as a stack to analyze several data dimensions at once.



Bullet Dialog

Properties for the Bullet are available in the Bullet dialog. To open it, with the Bullet control selected on the report, under the Properties Window, click the **Property dialog** link.

The Bullet dialog lets you set properties on the report control with the following pages.

> **Note:** You can click <**Expression...**> in many of these properties to open the Expression Editor where you can create an expression to determine the value. For properties with enumerated values, the values are listed under **Constants** in the **Fields** tree view on the left side of the Expression Editor.

General

Name: Enter a name for the Bullet that is unique within the report. You can only use underscore (_) as a special character in the Name field. Other special characters such as period (.), space ( ), forward slash (/), back slash (\), exclamation (!), and hyphen (-) are not supported.

Data

Value: Enter an expression to use as the bullet value.

Target Value: Enter an expression to use as the target value of the bullet graph.

Appearance

Bullet Graph Orientation

Horizontal: Select to display a horizontal bullet graph.

Vertical: Select to display a vertical bullet graph.

Value Style

Color: Select a color to use for the value marker, or select the <**Expression...**> option to open the Expression Editor and create an expression that evaluates to a .NET color. The default value is **Black**.

Target Style

Target Type: Choose **Line**, **Dot** or **Square**. The default value is **Line**.

Color: Select a color to use for the target value marker, or select the <**Expression...**> option to open the Expression Editor and create an expression that evaluates to a .NET color. The default value is **Black**.

Width: Enter a value in points to set the width of the target value marker. The default value is **3pt**.

> **Note:** The **Width** setting applies only when the **Target Type** is set to **Line**.

Tick Marks

Position: Choose **None**, **Inside** or **Outside**. The default value is **Outside**.

Color: Select a color to use for the tick marks, or select the <**Expression...**> option to open the Expression Editor and create an expression that evaluates to a .NET color. The default value is **LightGray**.

Width: Enter a value in points to set the width of the tick marks. The default value is **1pt**.

Interval between tick marks: Set the interval at which you want to show tick marks.

Ranges

Worst Value: Enter a value or expression to define the lowest value on the graph.

Bad/Satisfactory Boundary: Enter a value or expression to define the boundary between bad and satisfactory values.

Display 3 Sections: Select this check box to show three separate value ranges (bad, satisfactory, and good) instead of two (bad and satisfactory). This enables the Satisfactory/Good Boundary.

Satisfactory/Good Boundary: Enter a value or expression to define the boundary between satisfactory and good values.

Best Value: Enter a value or expression to define the highest value on the graph.

Labels

Display Labels: Select this check box to display axis labels for the bullet graph. Selecting this box enables the rest of the properties on this page.

Format: Select one of the provided format codes or use a custom .NET formatting code to format dates or numbers. For more information, see MSDN's Formatting Types topic.

Font

Family: Choose the font family name. The default value is **Arial**.

Size: Choose the size in points for the font. The default value is **10pt**.

Style: Choose **Regular**, **Bold**, **Italic**, **Underline** or **Strikeout**. The default value is **Regular**.

Color: Select a Web or custom color for the font. The default value is **Black**.

Navigation

Document map label: Enter an expression to use as a label to represent this item in the table of contents (document map).

Bookmark ID: Enter an expression to use as a locator for this Bullet. You will then be able to provide a bookmark link to this item from another report control using a **Jump to bookmark** action.

Visibility

Initial visibility

- **Visible:** The bullet graph is visible when the report runs.
- **Hidden:** The bullet graph is hidden when the report runs.
- **Expression:** Use an expression with a Boolean result to decide whether the bullet graph is visible. True for hidden, False for visible.

Visibility can be toggled by another report control: Select this check box to display a toggle image next to another report control. This enables the drop-down box below where you can specify the TextBox control that toggles the visibility of the bullet. The user can click the toggle item to show or hide this bullet.

Data Output

Element Name: Enter a name to be used in the XML output for this Bullet.

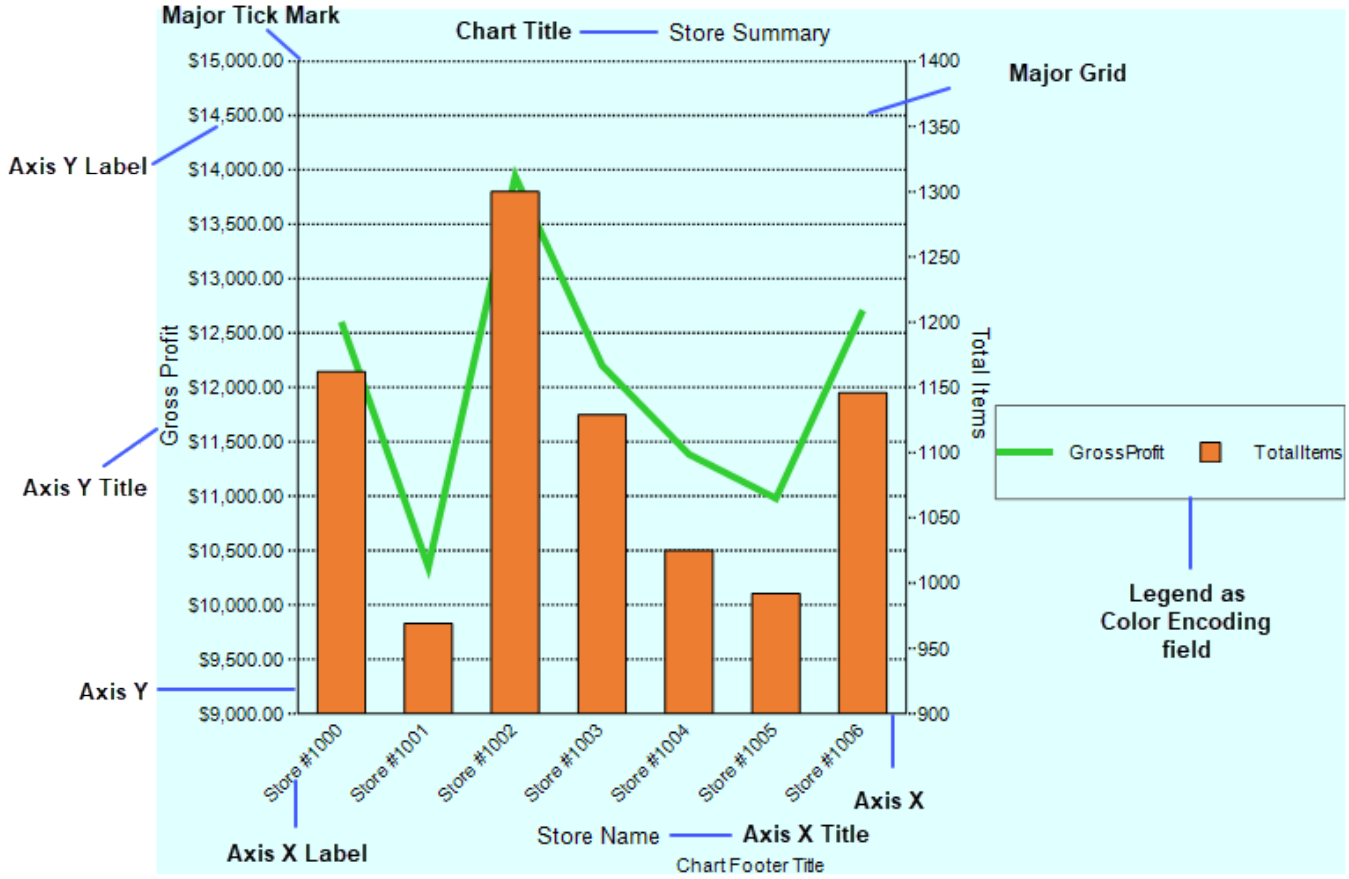
Output: Choose **Auto**, **Yes**, **No**, or **Content only** to decide whether to include this Bullet in the XML output. **Auto** exports the contents of the bullet graph only when the value is not a constant.

Chart

The Chart data region represents your data graphically. It is a data visualization control that is primarily based on slicing data through encodings.

Chart Elements

The Chart elements help you to easily analyze the visual information and interpret numerical and relational data. The following image illustrates the elements that make up the Chart data region.



Axis Label: A label along an axis that lets you label the units being shown.

Axis Title: The axis title allows you to provide a title for the information being shown on the axis.

Chart Title: The chart title serves as the title for the chart.

Grid Line: Grid lines can occur on horizontal and vertical axes and normally correlate to the major or minor tick marks for the axes.

Legend: The legend serves as a key to the specific colors or patterns being used to show series values in the chart. To display legends for each plot on a chart, use VALUESNAME as the color field encoding.

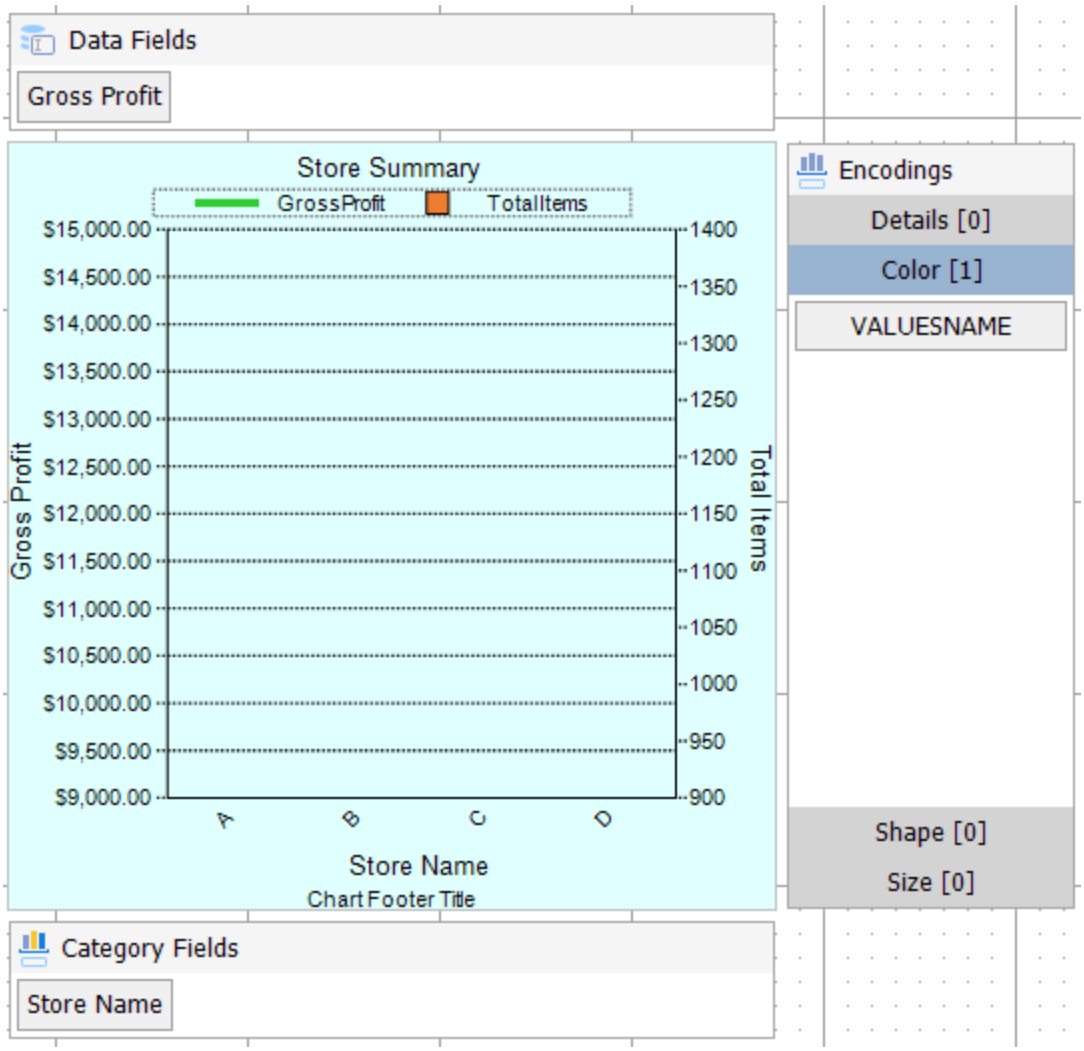
Major Tick: Major tick marks can occur on horizontal and vertical axes and normally correlate to the major gridlines for the axes.

Minor Tick: Minor tick marks can occur on horizontal and vertical axes and normally correlate to the minor gridlines for the axes.

Plot area: The Plot Area contains data plotted against X-axis and Y-axis.

Encodings

Encodings convert the data into visual elements. They define how each field from a data set is visualized. There are following encodings that are used in the process of creating charts.



Category Fields

The category mapping in Category Fields groups the data set into multiple categories for each unique value (or numerical range) that exists for the specified field.

Data Fields

The values mapping in Data Fields specifies which fields should be plotted along the Y-axis. For scatter plot charts, the values property includes mappings for both X and Y using comma delimited notation.

Details

The detail mapping slices the data into sub-divisions that can be clustered or stacked next to each other in the plot.

Color

The color mapping determines which field should be used to determine color assignations for the plot and legend.

Shape

The shape mapping determines which field should be used to determine shape assignations for the plot and legend.

Size

The size mapping determines which field to apply a continuous size-based scale to for the plot and legend.

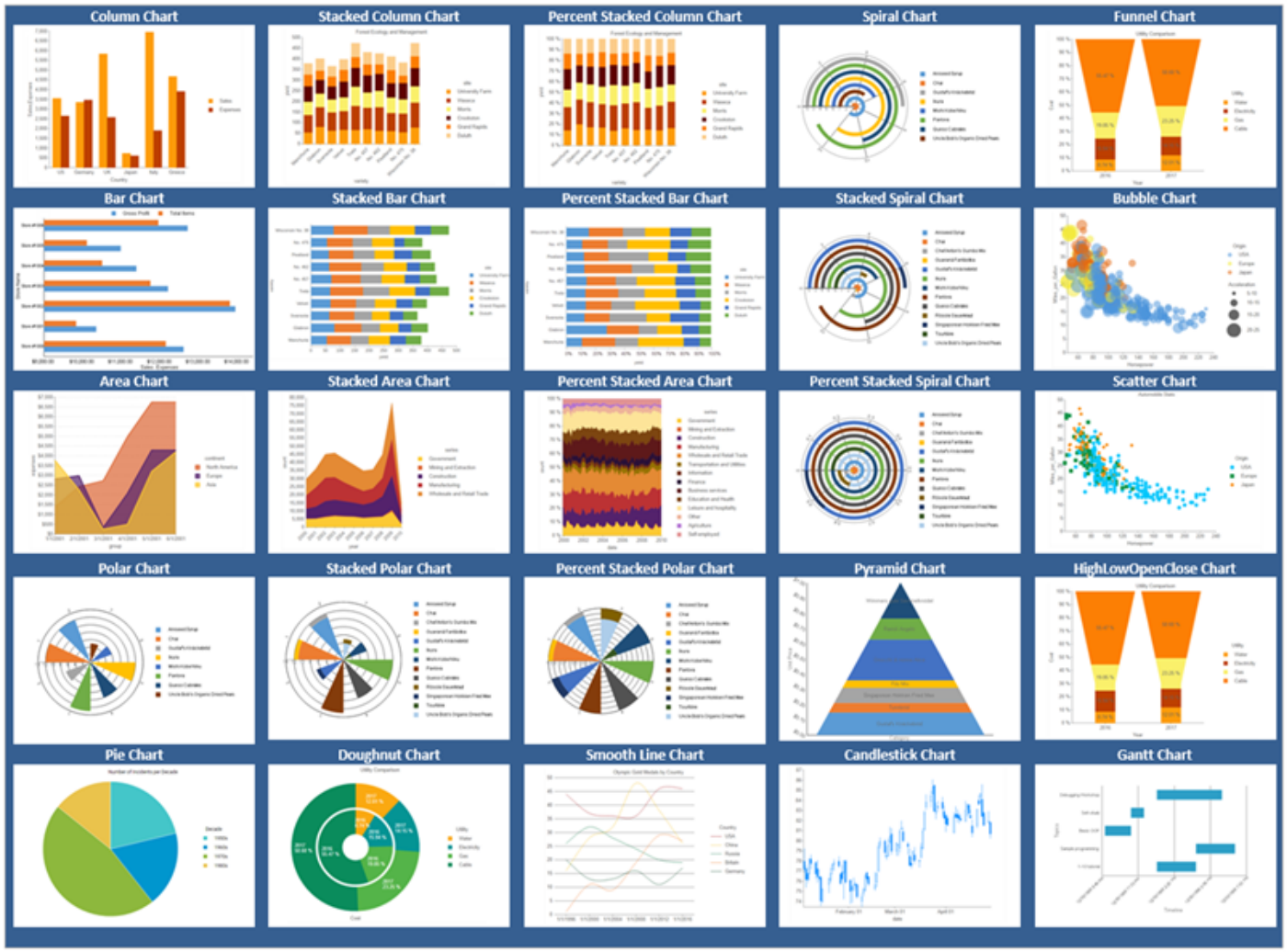
Important Properties

Property	Description

ToolTip	A textual label for the report item used to include Title or ALT attributes in HTML reports.
SwapAxes	Represents whether the axes are swapped.
Reversed	Represents whether the axis is reversed (top to bottom or right to left).
Padding	The amount of padding to place between the text or graphics and the edge of the report item.
LabelsAngle	Rotation angle of the axis labels. The angle is measured in degrees with valid values ranging from -90 to 90.
ShowNulls	Represents how to show the null or an empty value.
Sweep	Sweep of plots for radial plots.
InnerRadius	Inner radius in percentage for radial plots.
StartAngle	Starting angle of the radial plot.
Palette	The color support rgb and rgba format
Plots	The plots define the data fields that map to the chart. There can be single or multiple plots in a chart.
Tooltip Template	Contains the tooltip template settings. You can choose from the list of predefined settings or set your own in the Expression Editor.

Chart Types

Depending on the volume of data, number of variables, time period, and other data points, you can create charts from the following available types:



Column Chart
: Column, Stacked Column, and Percent Stacked Column.

Bar Chart
: Bar, Stacked Bar, and Percent Stacked Bar.

Line Chart
: Line and Smooth Line.

Area Chart
: Area, Stacked Area, and Stacked Percent Area.

Pie Chart
: Pie and Doughnut.

Spiral Chart
: Spiral, Stacked Spiral, and Percent Stacked Spiral.

Polar Chart
: Polar, Stacked Polar, and Percent Stacked Polar.

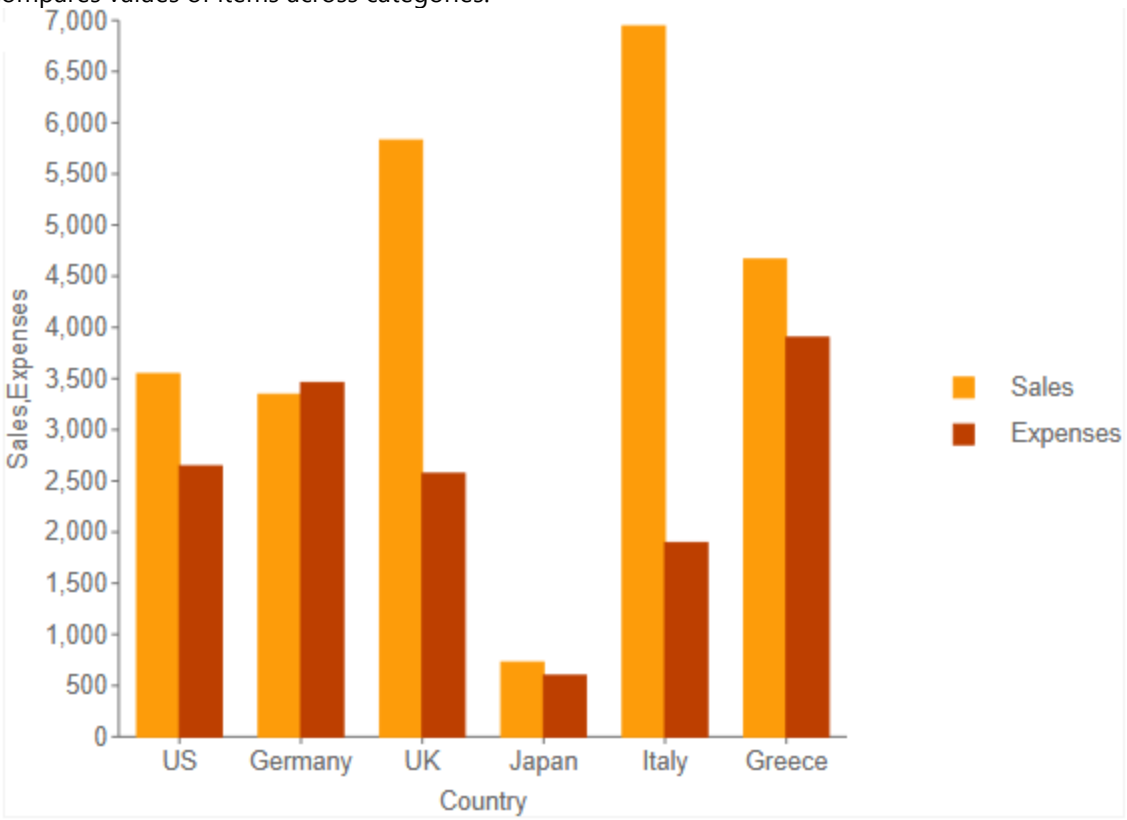
Other Chart Types
: Pyramid, Funnel, Bubble, Scatter, Gantt, High Low close, High Low Open Close, and Candlestick.

Column Chart

Column charts present each series as a vertical column, and group the columns by category. The y-axis values determine the heights of the columns, while the x-axis displays the category labels. With a column chart, you can select from the following subtypes.

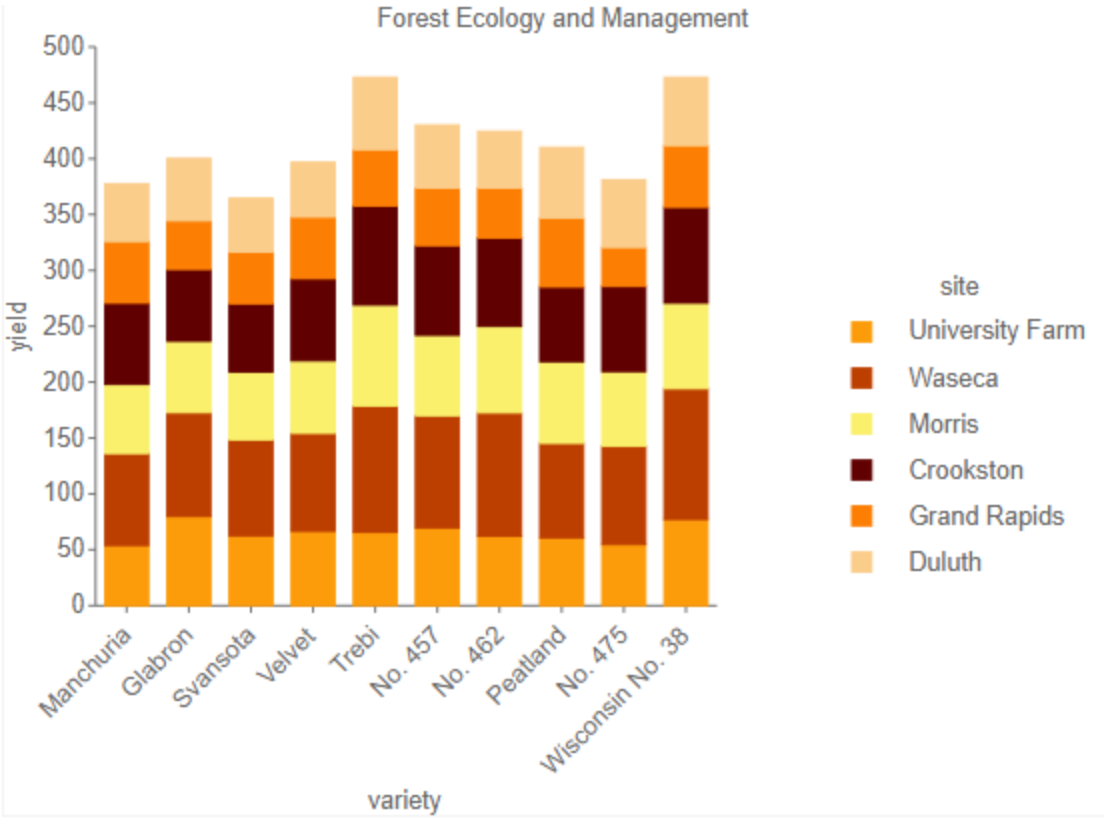
Column

Compares values of items across categories.



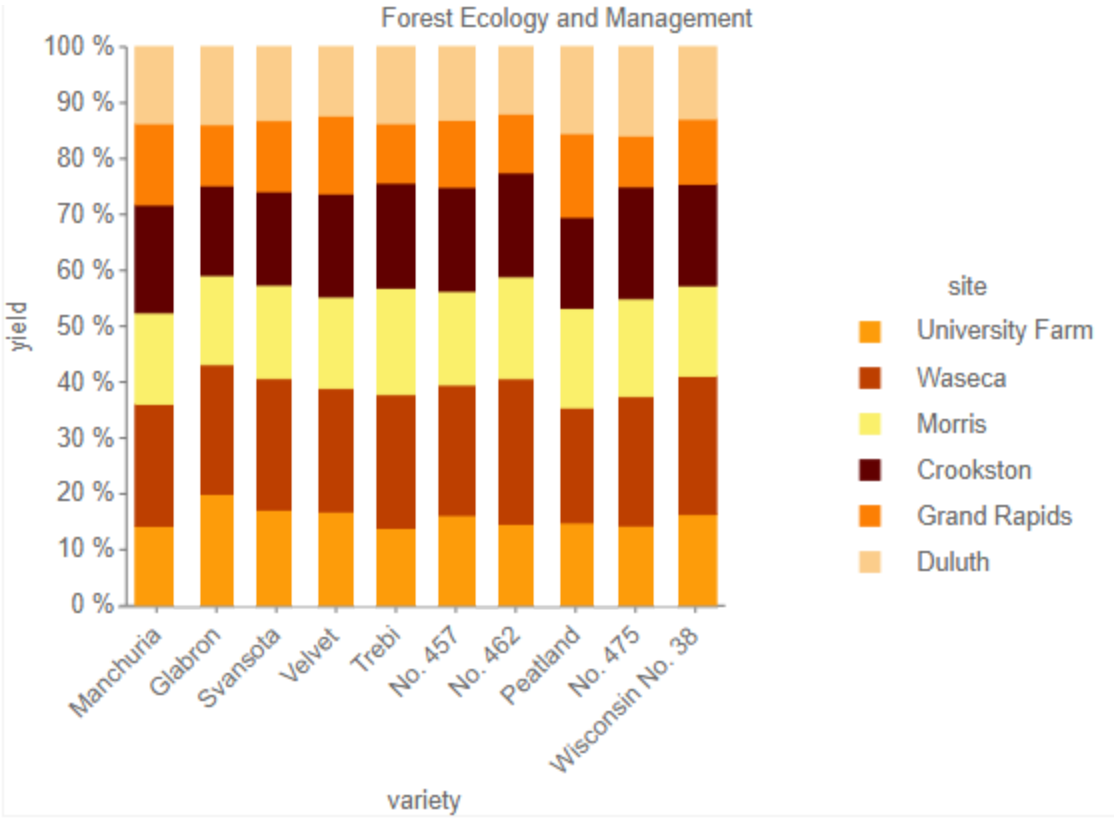
Stacked Column

A column chart with two or more data series stacked one on top of the other that shows how each value contributes to the total.



Percent Stacked Column

A column chart with two or more data series stacked one on top of the other to sum up to 100% that shows how each value contributes to a total with the relative size of each series representing its contribution to the total.

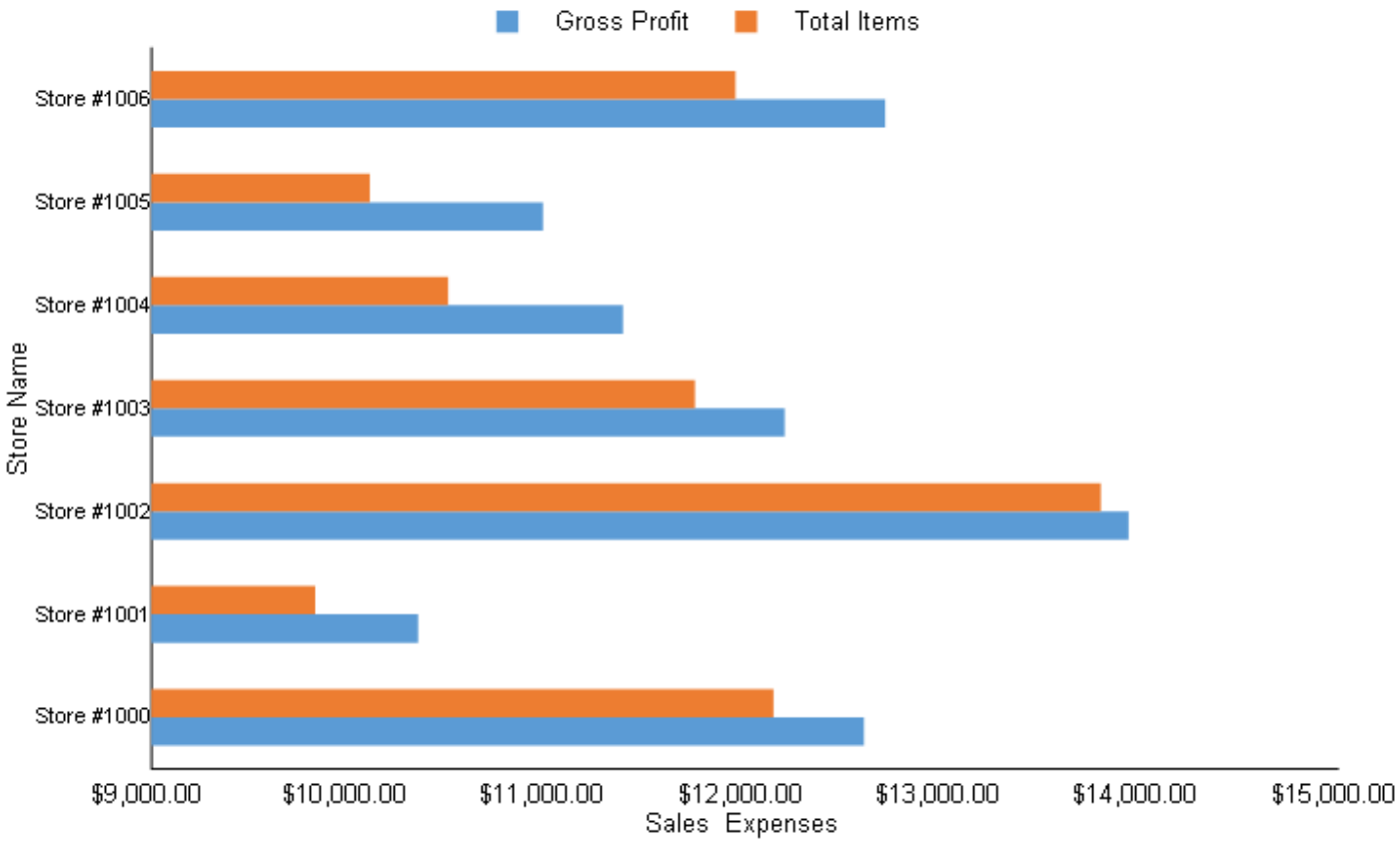


Bar Chart

Bar charts present each series as a horizontal bar, and group the bars by category. The x-axis values determine the lengths of the bars, while the y-axis displays the category labels. With a bar chart, you can select from the following subtypes.

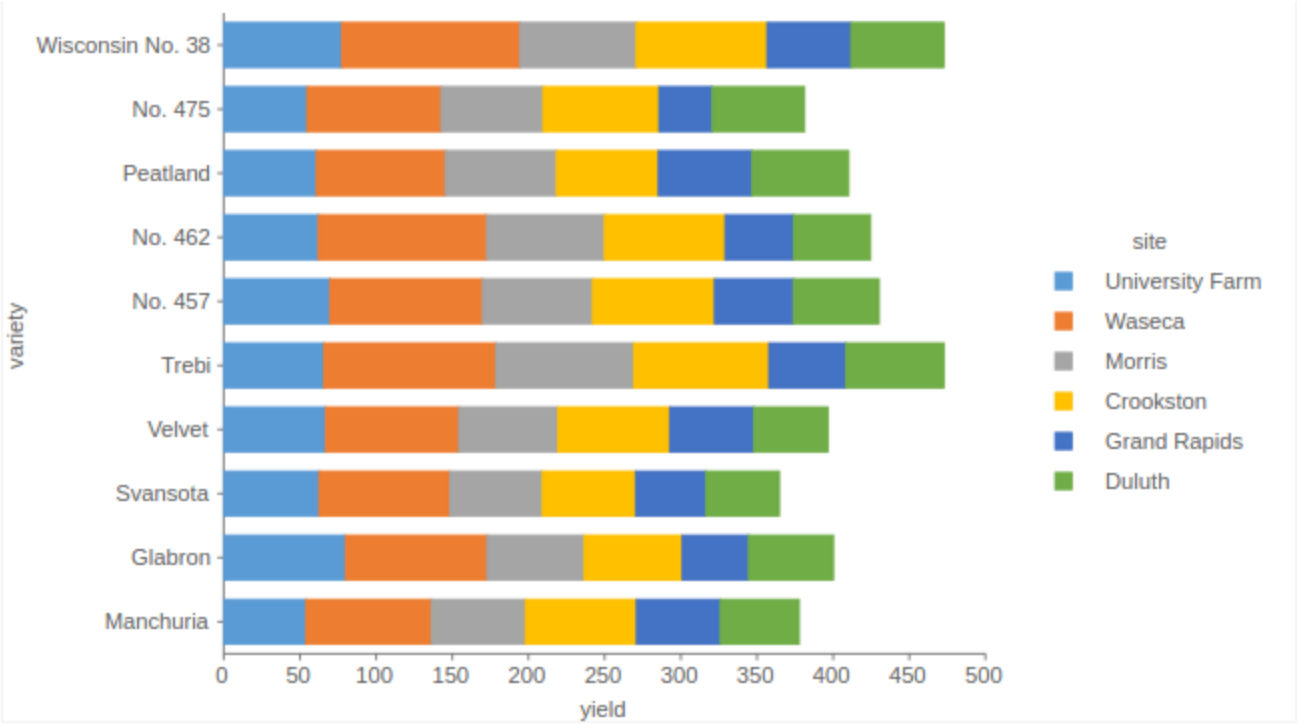
Bar

Compares values of items across categories.



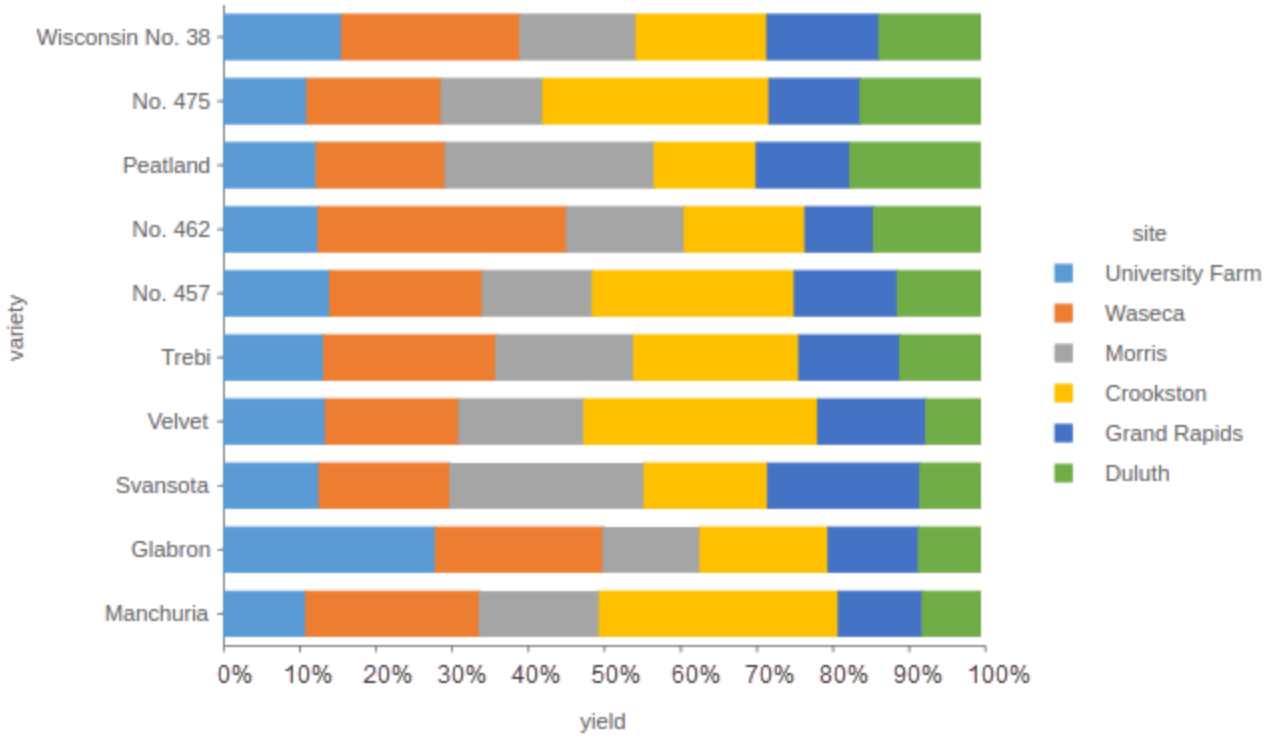
Stacked Bar

A bar chart with two or more data series stacked one on top of the other that shows how each value contributes to the total.



Percent Stacked Bar

A bar chart with two or more data series stacked one on top of the other to sum up to 100% that shows how each value contributes to the total with the relative size of each series representing its contribution to the total.

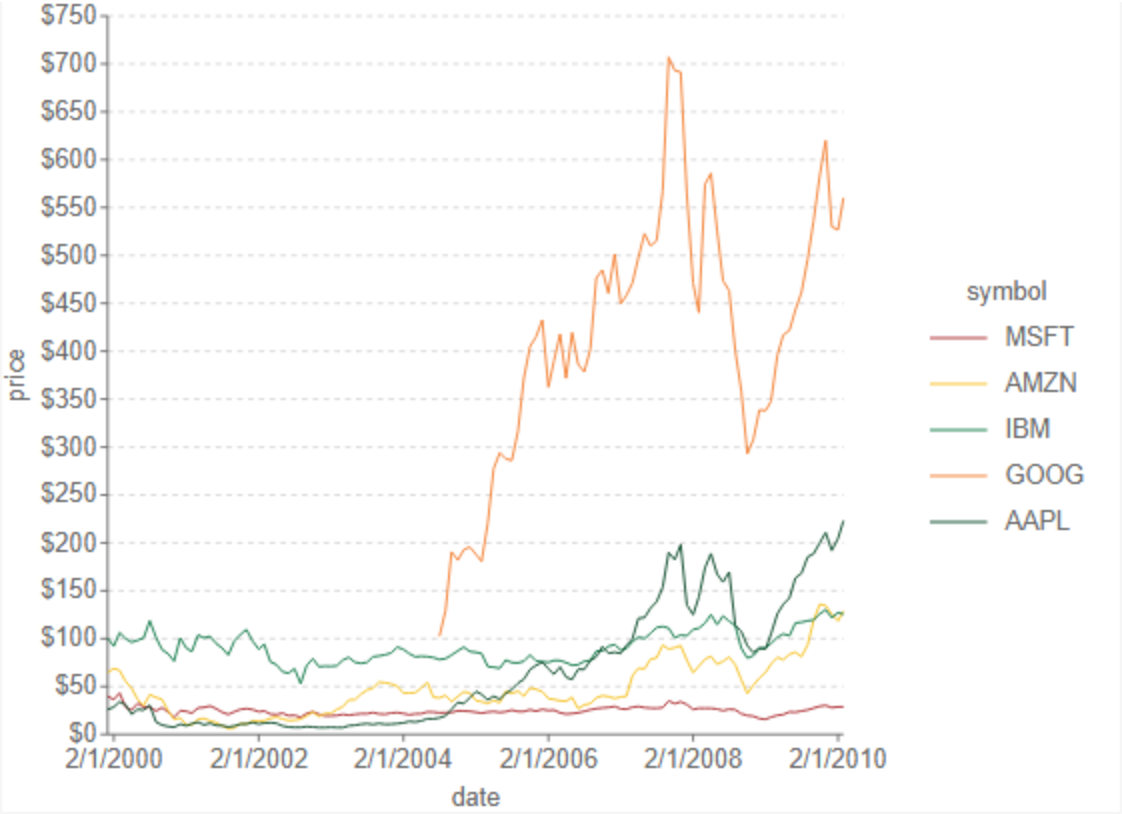


Line Chart

Line charts present each series as a point, and connect the points with a line. The y-axis values determine the heights of the points, while the x-axis displays the category labels. With a line chart, you can select from the following subtypes.

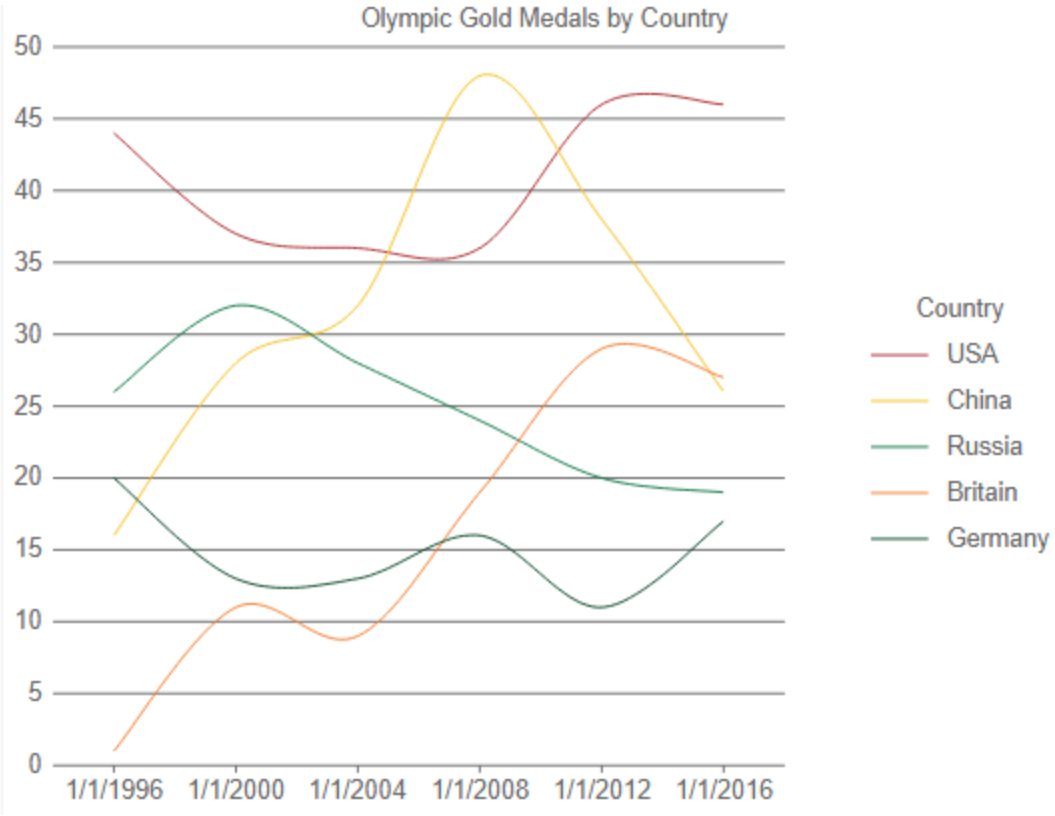
Line

Compares trends over a period of time or in certain categories.



Smooth Line

Plots curves rather than angled lines through the data points in a series to compare trends over a period of time or in certain categories.

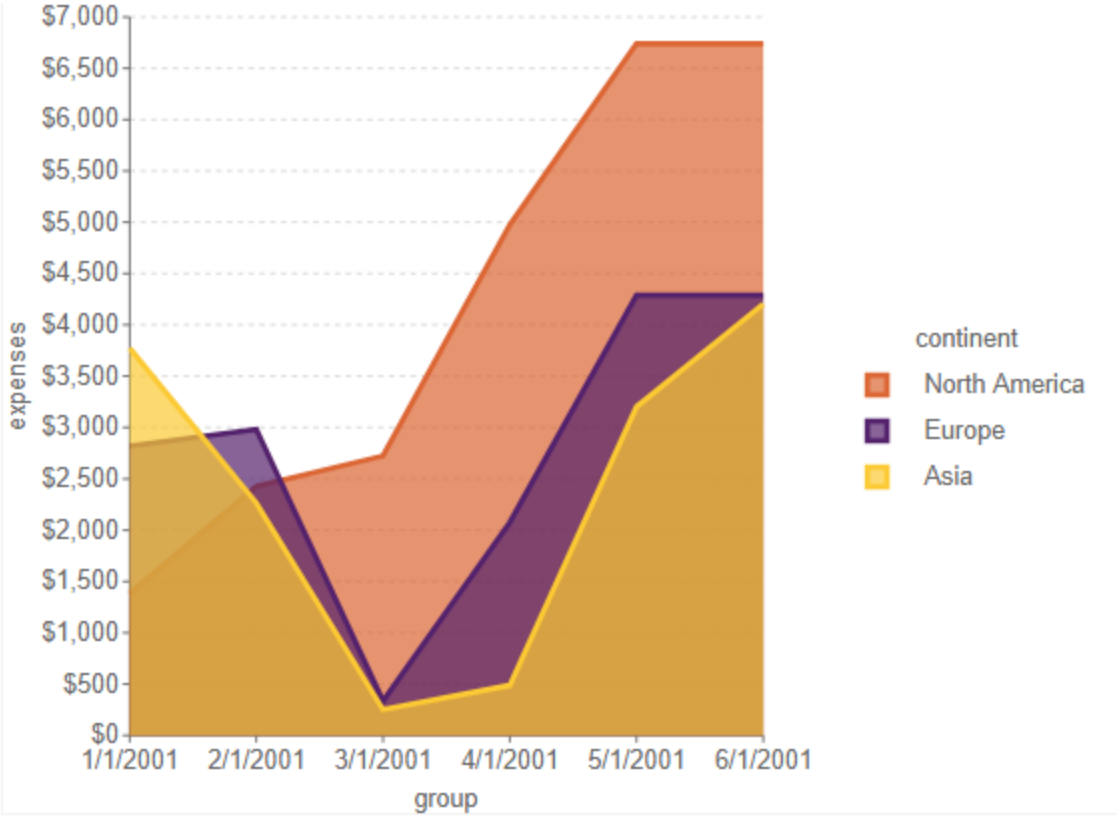


Area Chart

Area charts present each series as a point, connect the points with a line, and fill the area below the line. The y-axis values determine the heights of the points, while the x-axis displays the category labels. With an area chart, you can select from the following subtypes.

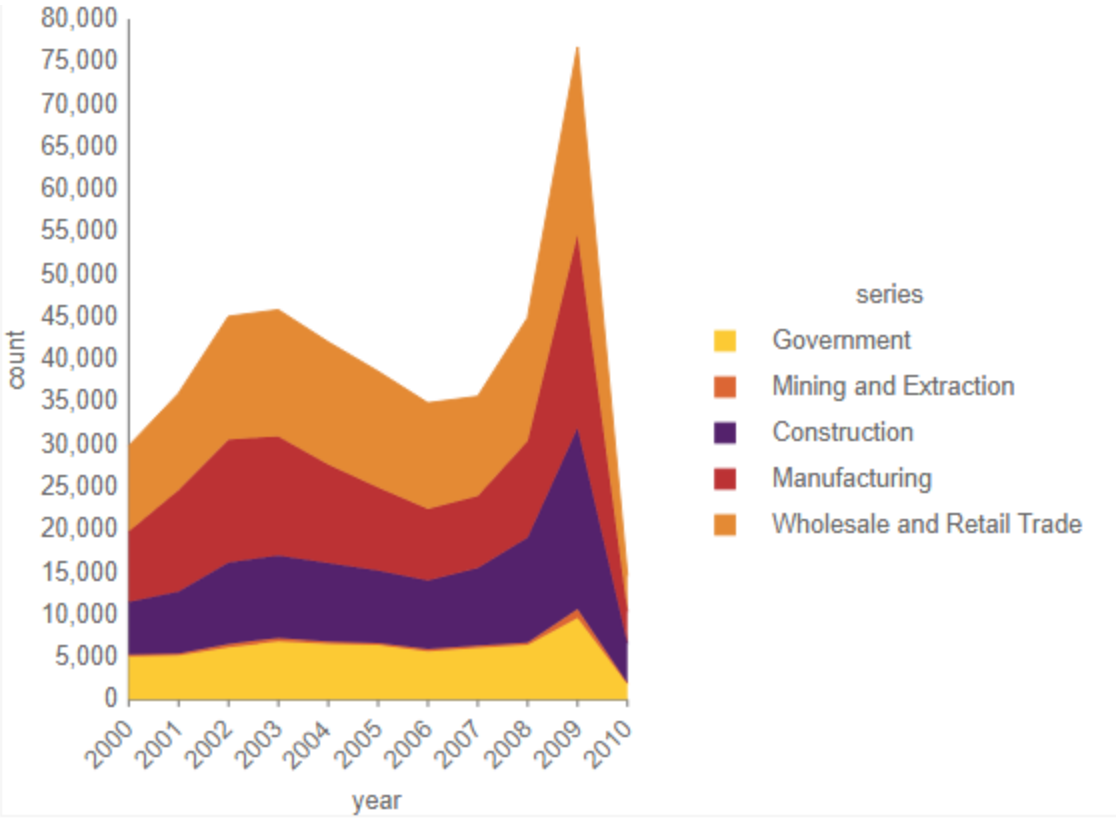
Area

Compare trends over a period of time or in specific categories.



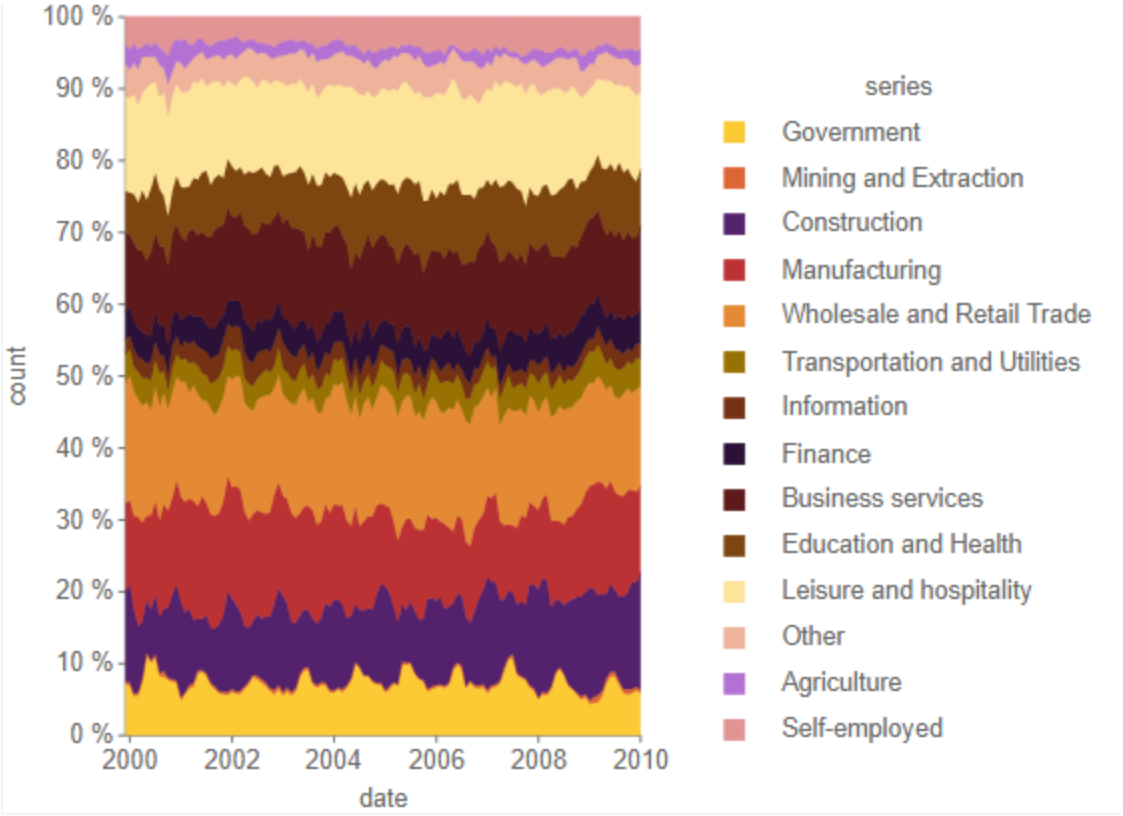
Stacked

An area chart with two or more data series stacked one on top of the other, shows how each value contributes to the total.



Percent Stacked

An area chart with two or more data series stacked one on top of the other to sum up to 100%, shows how each value contributes to the total with the relative size of each series representing its contribution to the total.

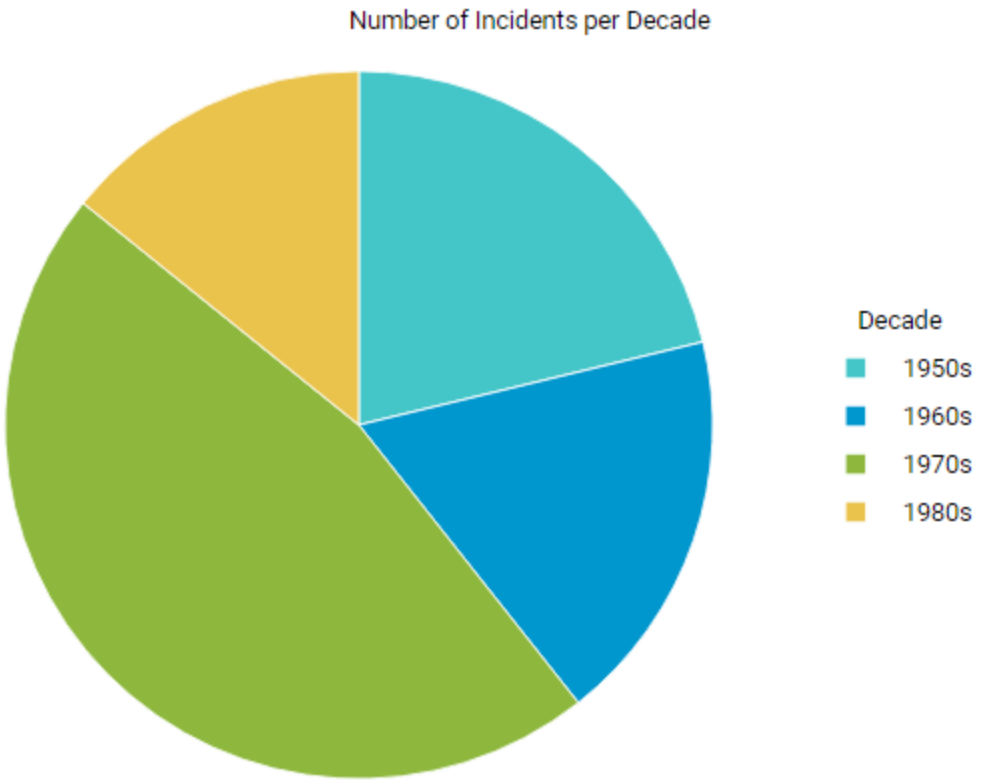


Pie Chart

Pie charts present each category as a slice of pie or doughnut, sized according to value. Series groups are not represented in pie charts. With a pie chart, you can select from the following subtypes.

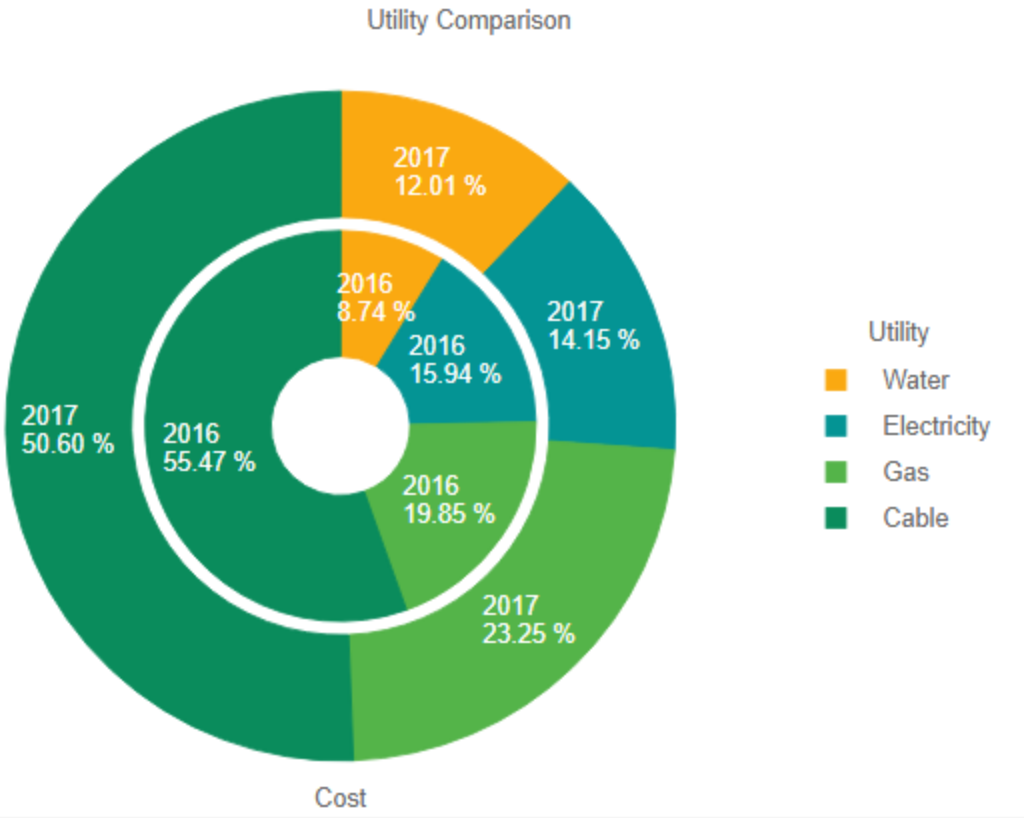
Pie

Shows how the percentage of each data item contributes to the total.



Doughnut

Shows how the percentage of each data item contributes to a total percentage.

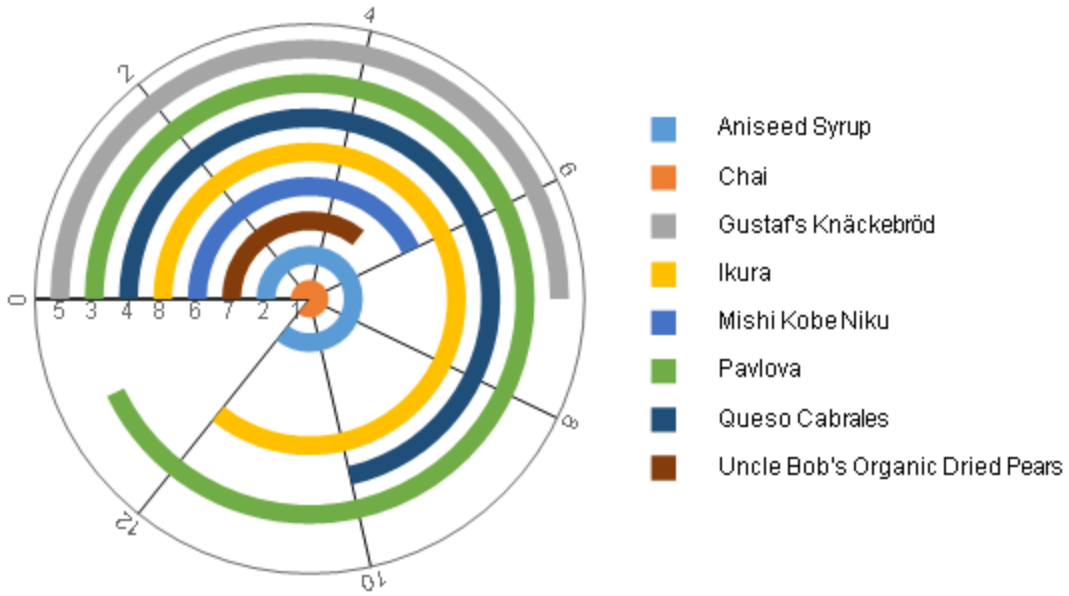


Spiral Chart

Spiral charts present data in the form of a spiral that is suitable for large data sets, showing trends over a long period of time. With a spiral chart, you can select from the following subtypes.

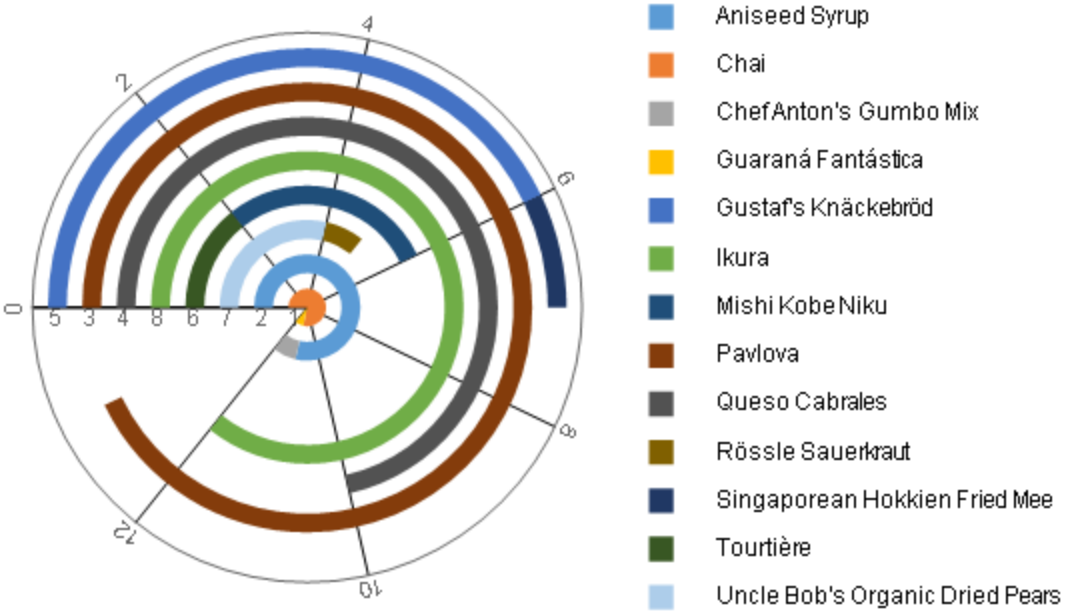
Spiral

Plots series along spirals, starting from the center of the circle.



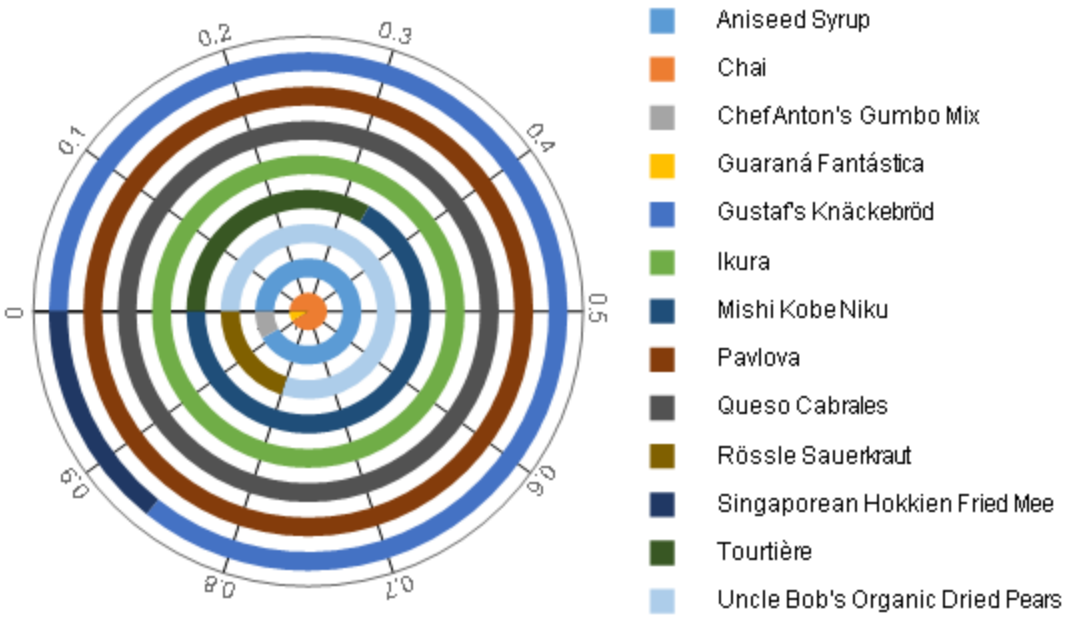
Stacked Spiral

Plots multiple series one on top of the other showing contribution of each value to the total.



Percent Stacked Spiral

Plots multiple series one on top of the other showing percentage contribution of each value to the total.

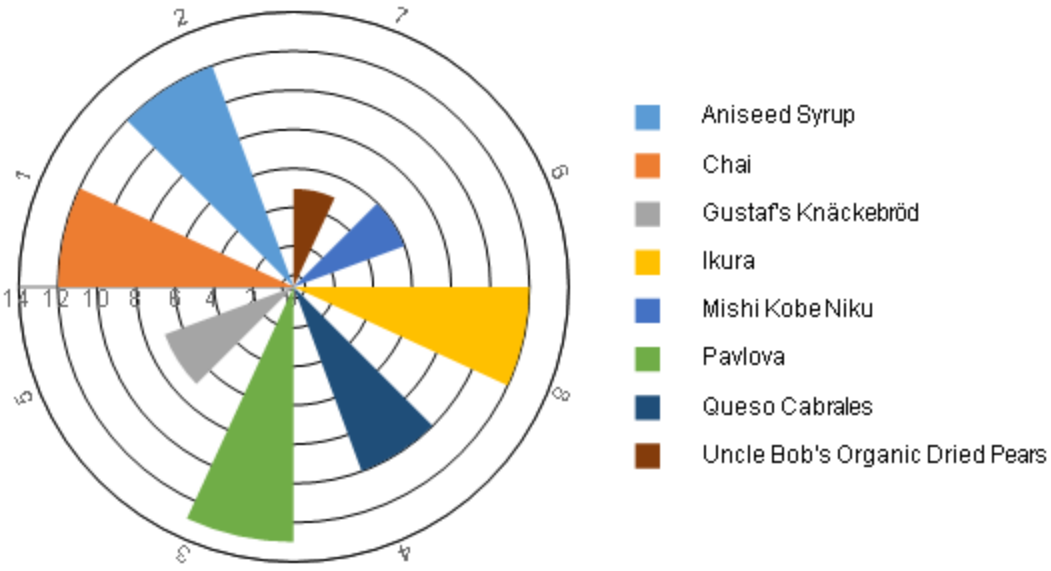


Polar Chart

Polar charts present data with geographical component. With a polar chart, you can select from the following subtypes.

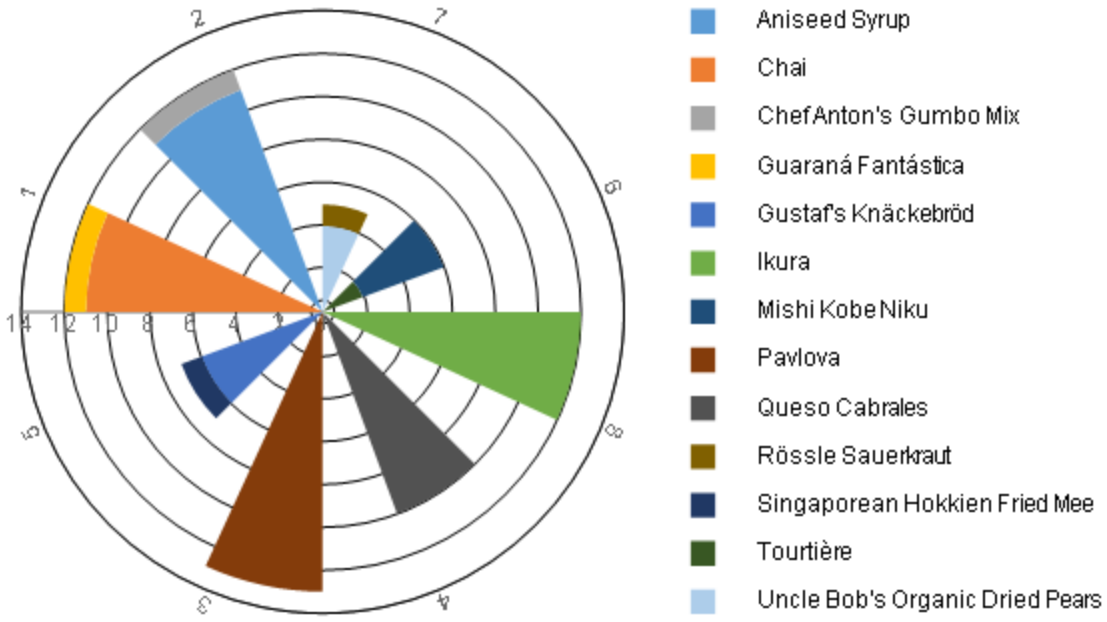
Polar

Polar charts plot series with circular x-axis representing angle values and radial y-axis representing radius values.



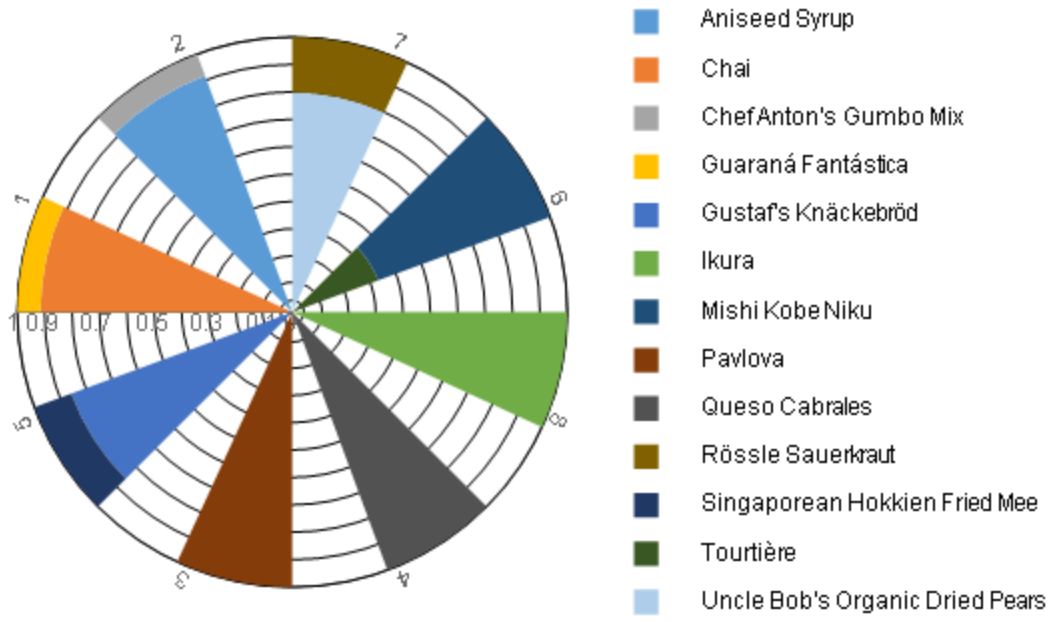
Stacked Polar

Plots multiple series one on top of the other showing contribution of each value to the total.



Percent Stacked Polar

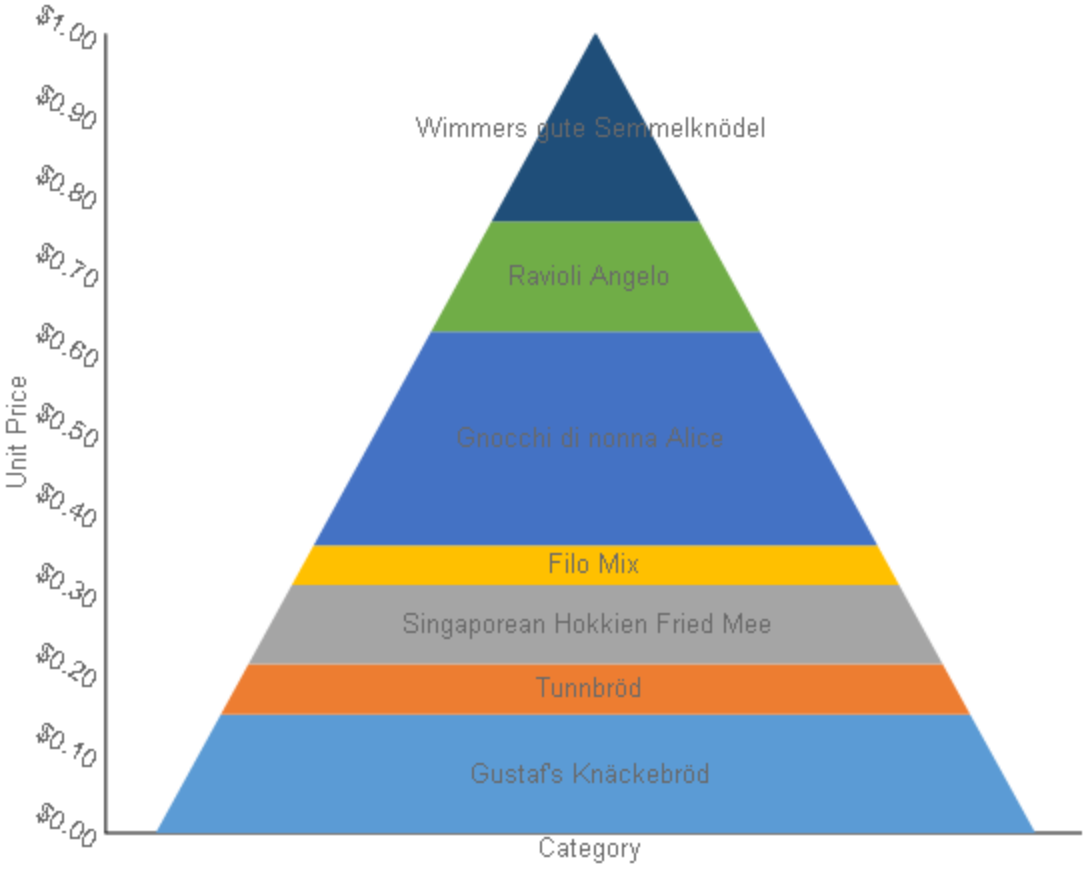
Plots multiple series one on top of the other showing percentage contribution of each value to the total.



Other Chart Types

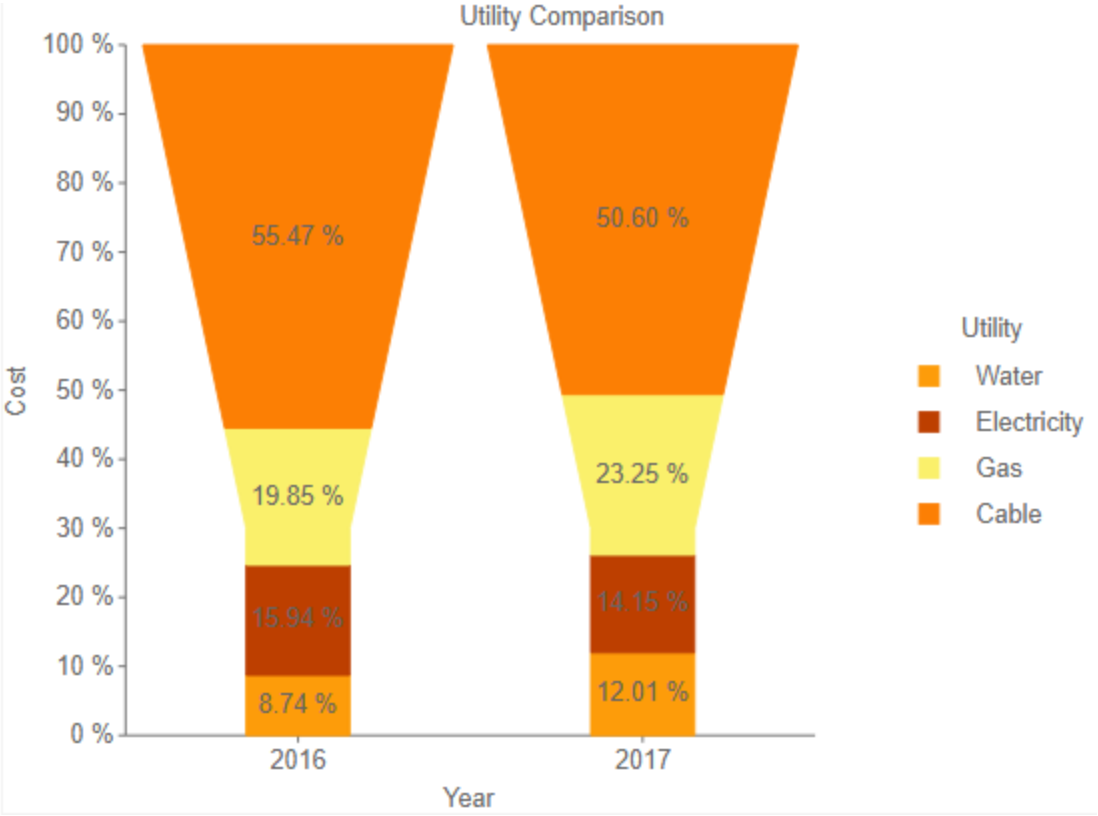
Pyramid

Shows how the percentage of each data item contributes to the whole, with the smallest value at the top and the largest at the bottom. This chart type works best with relatively few data items.



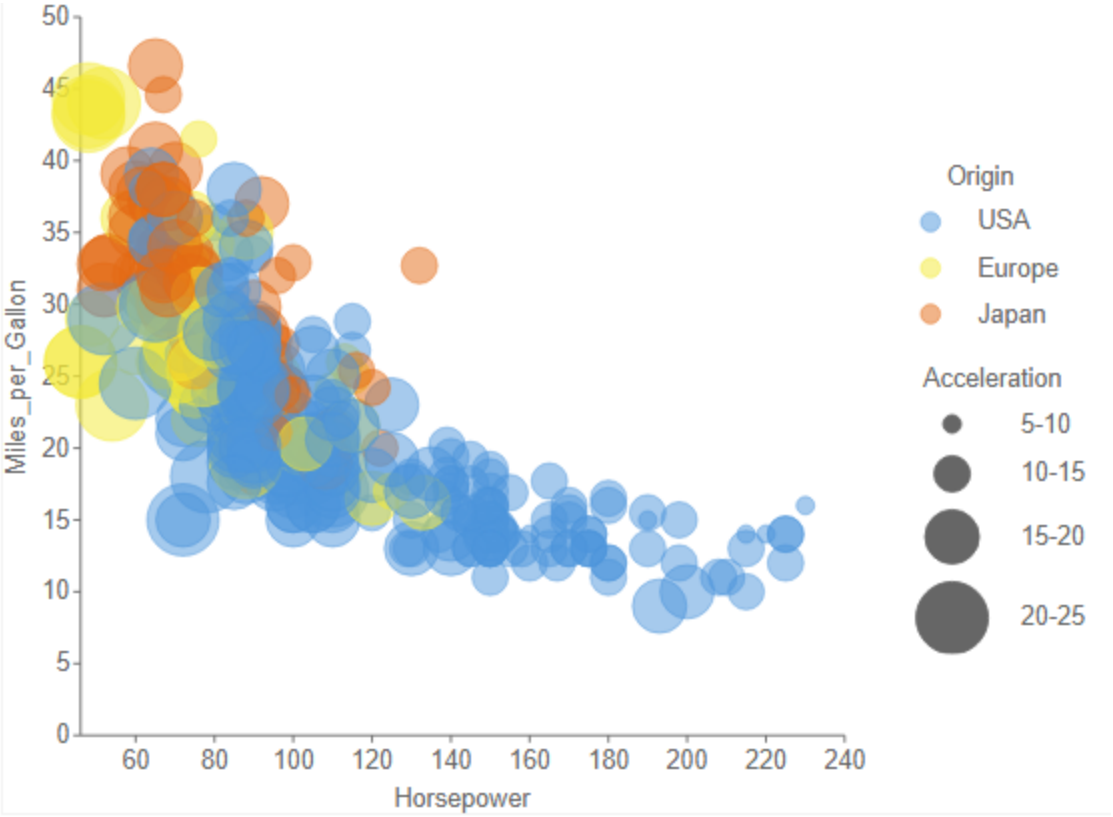
Funnel

Shows how the percentage of each data item contributes to the whole, with the largest value at the top and the smallest at the bottom. This chart type works best with relatively few data items.



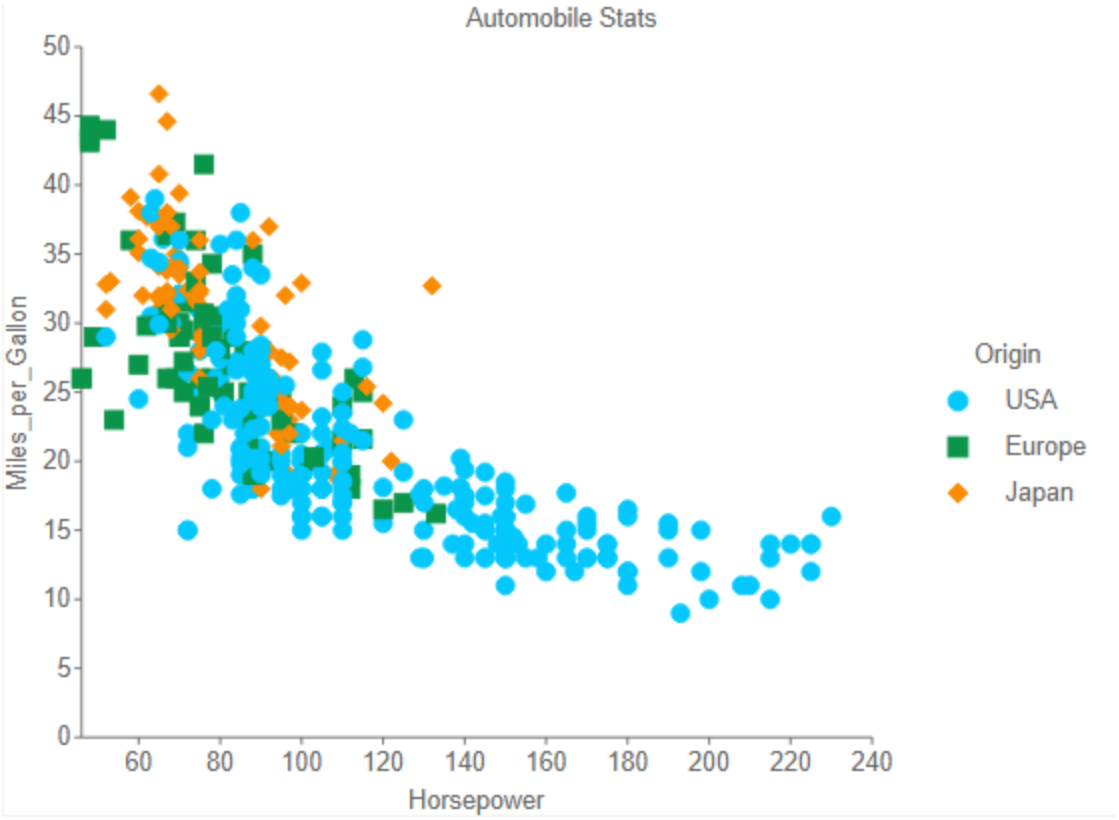
Bubble

Shows each series as a bubble. The y-axis values determine the height of the bubble, while the x-axis displays the category labels.



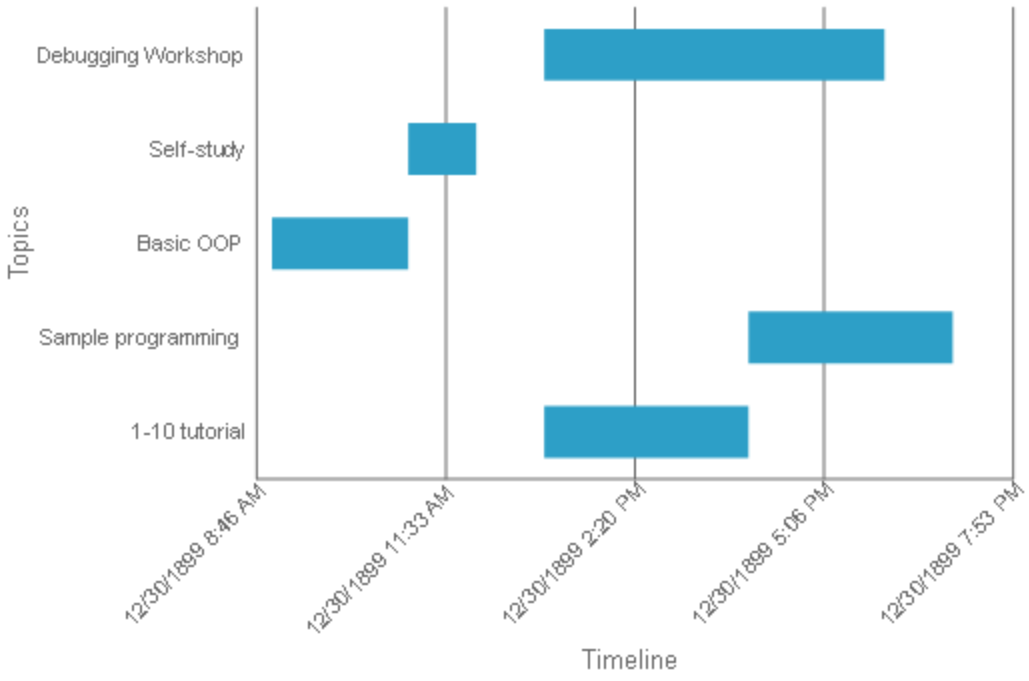
Scatter

Scatter charts present each series as a point or bubble. The y-axis values determine the heights of the points, while the x-axis displays the category labels. Shows the relationships between numeric values in two or more series sets of XY values.



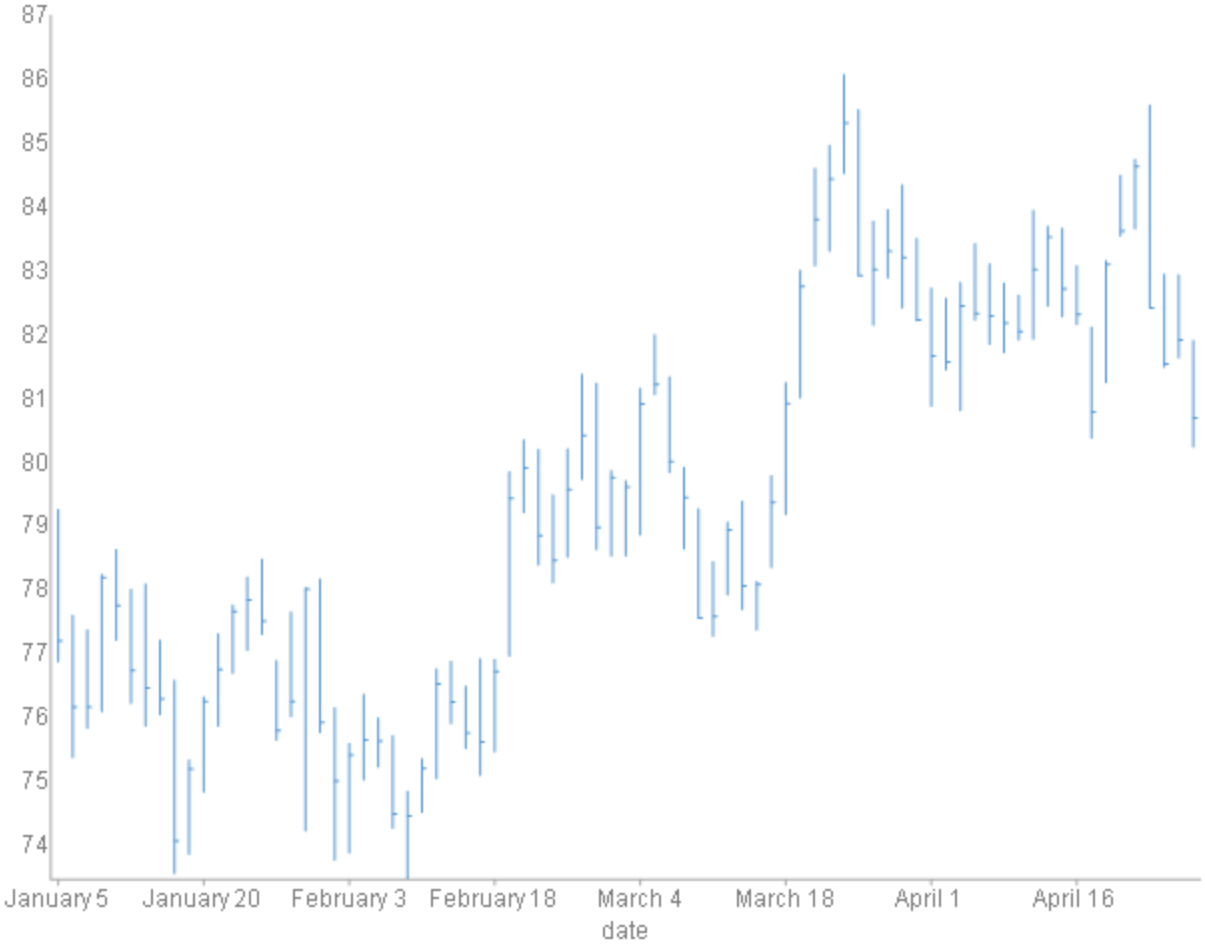
Gantt

This project management tool charts the progress of individual project tasks. The chart compares project task completion to the task schedule.



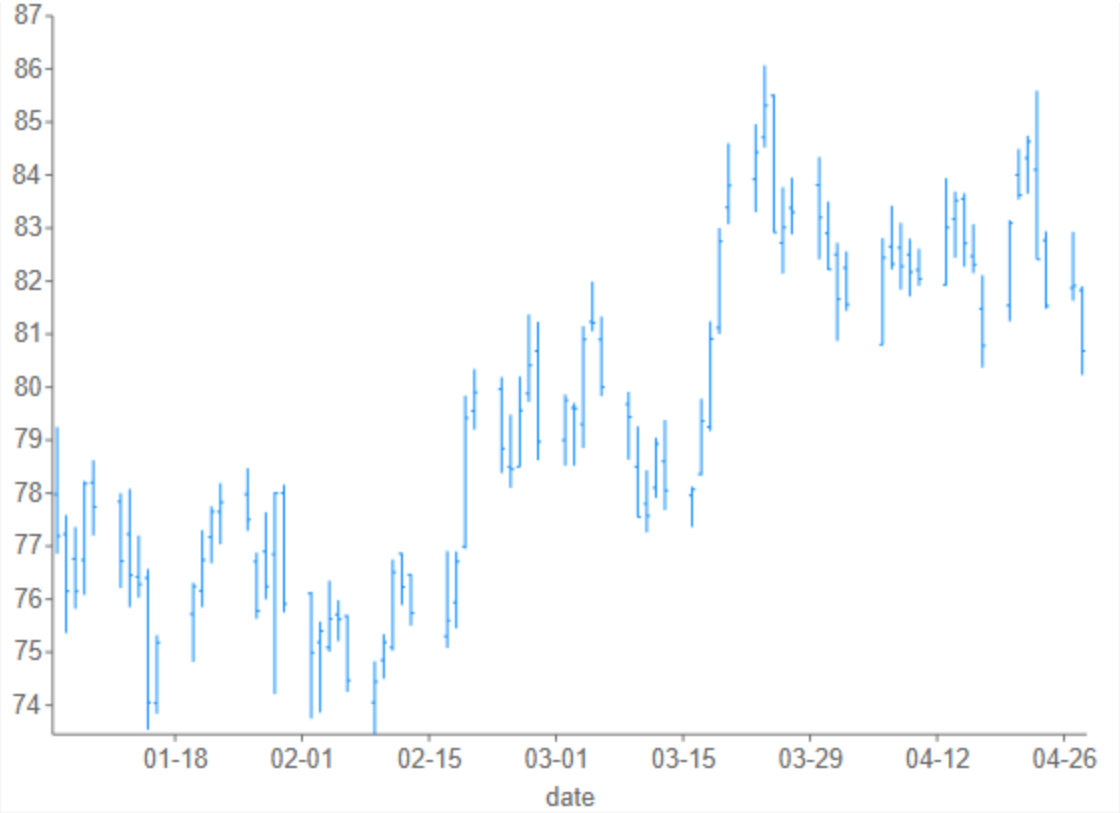
High Low Close

Displays stock information using High, Low, and Close values. High and low values are displayed using vertical lines, while tick marks on the right indicate closing values.



High Low Open Close

Displays stock information using High, Low, Open, and Close values. Opening values are displayed using lines to the left, while lines to the right indicate closing values. The high and low values determine the top and bottom points of the vertical lines.



Candlestick

Displays stock information using High, Low, Open and Close values. The height of the wick line is determined by the High and Low values, while the height of the bar is determined by the Open and Close values. The bar is displayed using different colors, depending on whether the price of the stock has gone up or down.

Classic Chart

> **Note**: The Classic Chart is by default hidden from the toolbox. If you want to use Classic Chart instead of new Chart, you can enable it from Grapecity.ActiveReports.config file located at C:\Program Files (x86)\GrapeCity\ActiveReports 14.

The **Chart** data region shows your data in a graphical representation that often makes it easier for users to comprehend large amounts of data quickly. Different types of charts are more efficient for different types of information, so we offer a wide variety of chart types. This makes it easy and cost effective to add charting to your reports, as there is no need to purchase and integrate a separate charting tool.

To hone in on your needs, when you first drag the Chart report control onto a Page report/RDL report, you can select the broad category of chart type to use: **Bar**, **Column**, **Scatter**, **Line**, or **Dot Plot**.

Once you select a chart category, there are a number of dialogs to help you to customize your chart.

> **Note**: You can select **<Expression...>** within many of these properties to create an expression to determine the value, or you can select a theme value to keep reports consistent.

Chart Appearance

To open the Chart Appearance dialog, select the Chart on the report, and below the Properties window, click the **Chart appearance** command. This dialog has the following pages.

> **Tip**: To go directly to the Plot Area page, click in the middle of the chart to select the **Plot Area**, then under the

> Properties Window, click **Property dialog**.

Gallery

The Gallery page of the Chart dialog, in basic mode, displays each of the broad categories of chart types, plus subtypes so that you can refine your choice. For even more chart types, click the **Advanced** button.

Basic Chart Types

Bar Charts

Bar charts present each series as a horizontal bar, and group the bars by category. The x-axis values determine the lengths of the bars, while the y-axis displays the category labels. With a bar chart, you can select from the following subtypes.

- **Plain:** Compares values of items across categories.
- **Stacked:** A bar chart with two or more data series stacked one on top of the other that shows how each value contributes to the total.
- **Percent Stacked:** A bar chart with two or more data series stacked one on top of the other to sum up to 100% that shows how each value contributes to the total with the relative size of each series representing its contribution to the total.

Column Charts

Column charts present each series as a vertical column, and group the columns by category. The y-axis values determine the heights of the columns, while the x-axis displays the category labels. With a column chart, you can select from the following subtypes.

- **Plain:** Compares values of items across categories.
- **Stacked:** A column chart with two or more data series stacked one on top of the other that shows how each value contributes to the total.
- **Percent Stacked:** A column chart with two or more data series stacked one on top of the other to sum up to 100% that shows how each value contributes to a total with the relative size of each series representing its contribution to the total.

Scatter Charts

Scatter charts present each series as a point or bubble. The y-axis values determine the heights of the points, while the x-axis displays the category labels. With a scatter chart, you can select from the following subtypes.

- **Plain:** Shows the relationships between numeric values in two or more series sets of XY values.
- **Connected:** Plots points on the X and Y axes as one series and uses a line to connect points to each other.
- **Smoothly Connected:** Plots points on the X and Y axes as one series and uses a line with the angles smoothed out to connect points to each other.
- **Bubble:** Shows each series as a bubble. The y-axis values determine the height of the bubble, while the x-axis displays the category labels. This chart type is only accessible in **Advanced** chart types.

Line Charts

Line charts present each series as a point, and connect the points with a line. The y-axis values determine the heights of the points, while the x-axis displays the category labels. With a line chart, you can select from the following subtypes.

- **Plain:** Compares trends over a period of time or in certain categories.
- **Smooth:** Plots curves rather than angled lines through the data points in a series to compare trends over a period of time or in certain categories. Also known as a Bezier chart.

Dot Plot Charts

A Dot Plot chart is a statistical chart containing group of data points plotted on a simple scale. Dot Plot chart are used for continuous, quantitative, univariate data. The dot plot chart has one subtype.

Plain: Displays simple statistical plots. It is ideal for small to moderate sized data sets. You can also highlight clusters and gaps, as well as outliers, while conserving numerical information.

Advanced Chart Types

Area Charts

Area charts present each series as a point, connect the points with a line, and fill the area below the line. The y-axis values determine the heights of the points, while the x-axis displays the category labels. With an area chart, you can select from the following subtypes.

- **Plain:** Compare trends over a period of time or in specific categories.
- **Stacked:** An area chart with two or more data series stacked one on top of the other, shows how each value contributes to the total.
- **Percent Stacked:** An area chart with two or more data series stacked one on top of the other to sum up to 100%, shows how each value contributes to the total with the relative size of each series representing its contribution to the total.

Pie Charts

Pie charts present each category as a slice of pie or doughnut, sized according to value. Series groups are not represented in pie charts. With a pie chart, you can select from the following subtypes.

- **Pie:** Shows how the percentage of each data item contributes to the total.
- **Exploded:** Shows how the percentage of each data item contributes to the total, with the pie slices pulled out from the center to show detail.
- **Doughnut:** Shows how the percentage of each data item contributes to a total percentage.
- **Exploded Doughnut:** Shows how the percentage of each data item contributes to the total, with the pie slices pulled out from the center to show detail.

Financial Charts

Stock charts present each series as a line with markers showing some combination of high, low, open, and close values. The y-axis values determine the heights of the lines, while the x-axis displays the category labels. With a financial chart, you can select from the following subtypes.

- **High Low Close:** Displays stock information using High, Low, and Close values. High and low values are displayed

using vertical lines, while tick marks on the right indicate closing values.

- **Open High Low Close:** Displays stock information using Open, High, Low, and Close values. Opening values are displayed using lines to the left, while lines to the right indicate closing values. The high and low values determine the top and bottom points of the vertical lines.
- **Candlestick:** Displays stock information using High, Low, Open and Close values. The height of the wick line is determined by the High and Low values, while the height of the bar is determined by the Open and Close values. The bar is displayed using different colors, depending on whether the price of the stock has gone up or down.
- **Renko:** Bricks of uniform size chart price movement. When a price moves to a greater or lesser value than the preset BoxSize value required to draw a new brick, a new brick is drawn in the succeeding column. A change in box color and direction signifies a trend reversal.
- **Kagi:** Displays supply and demand trends using a sequence of linked vertical lines. The thickness and direction of the lines vary depending on the price movement. If closing prices go in the direction of the previous Kagi line, then that Kagi line is extended. However, if the closing price reverses by the preset reversal amount, a new Kagi line is charted in the next column in the opposite direction. Thin lines indicate that the price breaks the previous low (supply) while thick lines indicate that the price breaks the previous high (demand).
- **Point and Figure:** Stacked columns of Xs indicate that demand exceeds supply and columns of Os indicate that supply exceeds demand to define pricing trends. A new X or O is added to the chart if the price moves higher or lower than the BoxSize value you set. A new column is added when the price reverses to the level of the BoxSize value multiplied by the ReversalAmount you set. This calculation of pricing trends is best suited for long-term financial analysis.
- **Three Line Break:** Vertical boxes or lines illustrate price changes of an asset or market. The price in a three line break graph must break the prior high or low set in the NewLineBreak property in order to reverse the direction of the graph.

Other Charts

Other chart types may be used for special functions like charting the progress of individual tasks. You can select from the following subtypes.

- **Funnel:** Shows how the percentage of each data item contributes to the whole, with the largest value at the top and the smallest at the bottom. This chart type works best with relatively few data items.
- **Pyramid:** Shows how the percentage of each data item contributes to the whole, with the smallest value at the top and the largest at the bottom. This chart type works best with relatively few data items.
- **Gantt:** This project management tool charts the progress of individual project tasks. The chart compares project task completion to the task schedule.

Title

Chart title: Enter an expression or text to use for the title.

Font

Family: Choose the font family name.

Size: Choose the size in points for the font.

Style: Choose Normal or Italic.

Weight: Choose from Lighter, Thin, ExtraLight, Light, Normal, Medium, SemiBold, Bold, ExtraBold, Heavy, and Bolder.

Color: Select a Web or custom color for the font.

Decoration: Choose from None, Underline, Overline, and LineThrough.

Palette

Default: The same as Subdued below, the recommended palette for charts.

EarthTones: A palette of autumnal browns, oranges, and greens.

Excel: A palette of muted plums, blues, and creams.

GrayScale: A palette of patterns suitable for printing to a black and white printer.

Light: A palette of pale pinks and peaches.

Pastel: A palette of blues, greens, and purples.

SemiTransparent: A palette of primary and tertiary colors that allows the backdrop to show through.

Subdued: A palette of muted tones of browns, greens, blues, and grays.

Vivid: The same as Subdued, but with richer tones.

Custom: A palette of colors that you define. When you select Custom, you can list colors that are used in the order you specify.

Area

Border

Style: Choose an enumerated style for the border.

Width: Set a width value in points between **0.25pt** and **20pt**.

Color: Select a Web or Custom color.

Background Fill Color

Fill Color: Select a Web or Custom color.

Gradient: Choose from one of the following gradient styles.

- **None**: No gradient is used. The Fill Color is used to fill the area and the Gradient End Color property is ignored.
- **LeftRight**: A gradient is used. The Fill Color property defines the color at the left, and the Gradient End Color property defines the color at the right. The two colors are gradually blended in between these areas.
- **TopBottom**: A gradient is used. The Fill Color property defines the color at the top, and the Gradient End Color property defines the color at the bottom. The two colors are gradually blended in between these areas.
- **Center**: A gradient is used. The Fill Color property defines the color at the center, and the Gradient End Color property defines the color at the edges. The two colors are gradually blended in between these areas.
- **DiagonalLeft**: A gradient is used. The Fill Color property defines the color at the top left, and the Gradient End Color property defines the color at the bottom right. The two colors are gradually blended in between these areas.
- **DiagonalRight**: A gradient is used. The Fill Color property defines the color at the top right, and the Gradient End Color property defines the color at the bottom left. The two colors are gradually blended in between these areas.
- **HorizontalCenter**: A gradient is used. The Gradient End Color property defines the horizontal band of color across the center, and the Fill Color property defines the color at the top and bottom. The two colors are gradually blended in between these areas.
- **VerticalCenter**: A gradient is used. The Gradient End Color property defines the vertical band of color across the center, and the Fill Color property defines the color at the left and right. The two colors are gradually blended in between these areas.

Gradient End Color: When you choose any gradient style other than None, this property becomes available. Choose a Web or Custom color.

Plot Area

Border

Style: Choose an enumerated style for the border.

Width: Choose a width value between **0.25pt** and **20pt**.

Color: Select a Web or Custom color.

Background Fill Color

Fill Color: Select a Web or Custom color.

Gradient: Choose from one of the following gradient styles.

- **None**: No gradient is used. The Fill Color is used to fill the area and the Gradient End Color property is ignored.
- **LeftRight**: A gradient is used. The Fill Color property defines the color at the left, and the Gradient End Color property defines the color at the right. The two colors are gradually blended in between these areas.
- **TopBottom**: A gradient is used. The Fill Color property defines the color at the top, and the Gradient End Color property defines the color at the bottom. The two colors are gradually blended in between these areas.
- **Center**: A gradient is used. The Fill Color property defines the color at the center, and the Gradient End Color property defines the color at the edges. The two colors are gradually blended in between these areas.
- **DiagonalLeft**: A gradient is used. The Fill Color property defines the color at the top left, and the Gradient End Color property defines the color at the bottom right. The two colors are gradually blended in between these areas.
- **DiagonalRight**: A gradient is used. The Fill Color property defines the color at the top right, and the Gradient End Color property defines the color at the bottom left. The two colors are gradually blended in between these areas.
- **HorizontalCenter**: A gradient is used. The Gradient End Color property defines the horizontal band of color across the center, and the Fill Color property defines the color at the top and bottom. The two colors are gradually blended in between these areas.
- **VerticalCenter**: A gradient is used. The Gradient End Color property defines the vertical band of color across the center, and the Fill Color property defines the color at the left and right. The two colors are gradually blended in between these areas.

Gradient End Color: When you choose any gradient style other than None, this property becomes available. Choose a Web or Custom color.

Series line (Bar Stacked and Percent Stacked, Column Stacked and Percent Stacked types only)

Style: Choose an enumerated style for the series line.

Width: Choose a width value between **0.25pt** and **20pt**.

Color: Select a Web or Custom color.

3D Effects

These properties are enabled when you select the **3D** checkbox on the Gallery page.

Display the chart with 3D visual effects: Select this check box to enable all of the following properties.

Horizontal rotation: Move the slider to rotate the chart to left and right. All the way to the left (-90°) shows the chart from the left side, while all the way to the right (90°) shows it from the right side. The default value is 20°.

Vertical rotation: Move the slider to rotate the chart up and down. All the way to the left (-90°) shows the chart from the bottom, while all the way to the right (90°) shows it from the top. The default value is 20°.

Wall thickness: Move the slider to change the thickness of the walls at the axes. The default value is 0% and the range of values is 0% (left) to 100% (right). If the chart type is pie or doughnut, this property is ignored.

Perspective: Move the slider to change the perspective from which the chart is displayed. The default value is 0% and the range of values is 0% (left) to 100% (right). If you select Orthographic Projection, this property is ignored.

Shading: Select the type of shading to apply to the chart. The default value is Real.

- **None**: Colors are uniform.
- **Simple**: Colors are darkened in areas where the light source does not hit them.
- **Real**: Colors are darkened in areas where the light source does not hit them, and lightened in areas where the light source is strongest.

Orthographic Projection: Select this check box to use orthographic or "true drawing" projection. This type of projection is ignored with pie and doughnut chart types.

Clustered: With chart types of bar and column, select this check box to cluster series groups. Other chart types ignore this setting.

Display bars as cylinders: With chart types of bar and column, select this check box to display cylinders instead of bars or columns.

Defaults (button): Click this button when you want to set all of the 3D effect properties back to their default values.

Chart Data

See the Chart Data Dialog topic for all of the pages and tabs available for customizing your chart data.

Chart Legend

To open the Chart Legend dialog, select the Chart on the report, and below the Properties window, click the **Chart legend** command. This dialog has the following pages.

General

Show chart legend: Clear this check box to disable the legend. This also disables all of the other properties on this page.

Use Smart Settings: Check this option to apply smart settings or clear this checkbox to activate the properties given below.

Layout: Choose the layout style for the legend.

- **Column**: This option displays legend items in a single vertical column.
- **Row**: This option displays legend items in a single horizontal row.
- **Table**: This option displays legend items in a table of vertical columns, and is best when you have a large number of values.

Position: Select an enumerated value to determine the position of the legend relative to the chart area. The default value is **RightCenter**.

Display legend inside plot area: Select this check box to display the legend inside the plot area along with your data

elements.

Style

The Style page of the Chart Legend dialog allows you to control the Font, Border, and Fill properties for the legend.

Font

Family: Choose the font family name.

Size: Choose the size in points for the font.

Style: Choose Normal or Italic.

Weight: Choose from Lighter, Thin, ExtraLight, Light, Normal, Medium, SemiBold, Bold, ExtraBold, Heavy, and Bolder.

Color: Select a Web or custom color for the font.

Decoration: Choose from None, Underline, Overline, and LineThrough.

Border

Style: Choose an enumerated style for the border.

Width: Enter a width value between **0.25pt** and **20pt**.

Color: Select a Web or Custom color.

Background Fill Color

Fill Color: Select a Web or Custom color.

Gradient: Choose from one of the following gradient styles.

- **None**: No gradient is used. The Fill Color is used to fill the area and the Gradient End Color property is ignored.
- **LeftRight**: A gradient is used. The Fill Color property defines the color at the left, and the Gradient End Color property defines the color at the right. The two colors are gradually blended in between these areas.
- **TopBottom**: A gradient is used. The Fill Color property defines the color at the top, and the Gradient End Color property defines the color at the bottom. The two colors are gradually blended in between these areas.
- **Center**: A gradient is used. The Fill Color property defines the color at the center, and the Gradient End Color property defines the color at the edges. The two colors are gradually blended in between these areas.
- **DiagonalLeft**: A gradient is used. The Fill Color property defines the color at the top left, and the Gradient End Color property defines the color at the bottom right. The two colors are gradually blended in between these areas.
- **DiagonalRight**: A gradient is used. The Fill Color property defines the color at the top right, and the Gradient End Color property defines the color at the bottom left. The two colors are gradually blended in between these areas.
- **HorizontalCenter**: A gradient is used. The Gradient End Color property defines the horizontal band of color across the center, and the Fill Color property defines the color at the top and bottom. The two colors are gradually blended in between these areas.
- **VerticalCenter**: A gradient is used. The Gradient End Color property defines the vertical band of color across the center, and the Fill Color property defines the color at the left and right. The two colors are gradually blended in between these areas.

Gradient End Color: When you choose any gradient style other than None, this property becomes available. Choose a Web or Custom color.

Chart Axis

Click the Axis X or Axis Y line of the chart to select **AxisXLine** or **AxisYLine**, then under the Properties Window, click **Property dialog**. The Chart Axis dialogs let you set axis properties on the data region with the following pages.

> **Note**: The X and Y Axis dialogs are disabled if your chart type is doughnut or pie.

Title

Axis X or Axis Y

X- or Y-Axis title: Enter text to display near the X or Y axis of the chart.

Text alignment: Choose Center, Near, or Far.

Text orientation: Set the text orientation of the label text to Auto, Horizontal, Rotated90, Rotated270, and Stacked.

Font

Family: Choose the font family name.

Size: Choose the size in points for the font.

Style: Choose Normal or Italic.

Weight: Choose from Lighter, Thin, ExtraLight, Light, Normal, Medium, SemiBold, Bold, ExtraBold, Heavy, and Bolder.

Color: Select a Web or custom color for the font.

Decoration: Choose from None, Underline, Overline, and LineThrough.

The following are additional properties available for the Y-axis that allow you to plot two or more series along it. Composite charts have two or more series plotted along the Y-axis.

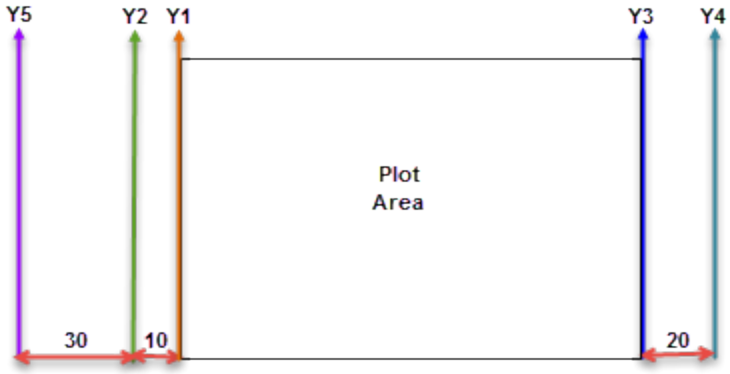
Name: Enter a name for each Y axis that is unique within the chart. Note that you can add a maximum of six Y-axes.

Position: Choose Left or Right.

Margin: Enter the distance between two consecutive Y-axes.

Name	Position	Margin	Description
Y1	Left	0	Displayed on the left. It is closest to the plot area.
Y2	Left	10	Displayed on the left of Y1. The distance between Y1 and Y2 is 10.
Y3	Right	0	Displayed on the right. It is closest to the plot area.
Y4	Right	20	Displayed on the right of Y3. The distance between Y3 and Y4 is 20.
Y5	Left	30	Displayed on the left of Y2. The distance between Y2 and Y5 is 30.

The image below depicts each of the Y-axes from the table above.



Line Style

Axis Line Appearance

Style: Choose from an enumerated style for the axis line.

Color: Select a Web or Custom color.

End Cap: Choose either None or Arrow as the End Cap style, or enter an expression using Expression Editor dialog.

Labels

Show x- or y-axis labels: Select this check box to show labels along the axis and to enable the rest of the properties on this page.

Format code: Select a format code from the list or use a custom .NET formatting code to format dates or numbers. For more information, see MSDN's Formatting Types topic.

Labels Text Orientation: Set the text orientation of the label text to Auto, Angled, Horizontal, Rotated90, Rotated270, and Stacked.

Font

Family: Choose the font family name.

Size: Choose the size in points for the font.

Style: Choose Normal or Italic.

Weight: Choose from Lighter, Thin, ExtraLight, Light, Normal, Medium, SemiBold, Bold, ExtraBold, Heavy, and Bolder.

Color: Select a Web or custom color for the font.

Decoration: Choose from None, Underline, Overline, and LineThrough.

Major Grid Lines

Show major grid lines: Select this check box to show grid lines for the axis.

Interval: Set the interval at which you want to show major grid lines or tick marks or both.

Border

Style: Choose one from the enumerated styles for the border.

Width: Enter a width value between **0.25pt** and **20pt**.

Color: Select a color for the border.

Tick mark: Choose one of the following values to determine whether and where to display major tick marks. The style and interval of the tick marks are set with the above properties.

- **None**: No tick mark is displayed.
- **Inside**: Tick marks are displayed inside the axis.
- **Outside**: Tick marks are displayed outside the axis.
- **Cross**: Tick marks are displayed crossing the axis.

Minor Grid Lines

Show minor grid lines: Select this check box to show minor grid lines for the axis.

Interval: Set the interval at which you want to show minor grid lines or tick marks or both.

Border

Style: Choose one from the enumerated styles for the border.

Width: Enter a width value between **0.25pt** and **20pt**.

Color: Select a color for the border.

Tick mark: Choose one of the following values to determine whether and where to display minor tick marks. The style and interval of the tick marks are set with the above properties.

- **None**: No tick mark is displayed.
- **Inside**: Tick marks are displayed inside the axis.
- **Outside**: Tick marks are displayed outside the axis.
- **Cross**: Tick marks are displayed crossing the axis.

Scale

Minimum: Leave this value blank to allow the data to determine the minimum value to use.

Maximum: Leave this value blank to allow the data to determine the maximum value to use.

Logarithmic scale: Select this check box to display axis data as a percentage of change instead of as absolute arithmetic values.

Numeric or time scale values: Select this check box to indicate that the data on the X axis is scalar so that the chart fills in missing numbers or dates between data values. This property is only available on the X axis.

Other

Cross at: Leave this value blank to allow the chart type to determine where the axis should cross the other axis, or you can enter a custom value.

Side margins: Select this check box to add padding between the data and the edges of the chart.

Interlaced strips: Select this check box to display alternating light and dark strips between major intervals specified on the Major Grid Lines page. If none are specified, a default value of 1 is used.

Reversed: Select this check box to reverse the direction of the chart. This will have different effects depending on chart type.

Reference Line (Y Axis only)

Value: Enter a value.

Line/Border

Style: Choose one from the enumerated styles.

Width: Set a width of the axis line.

Color: Select a color for the axis line.

Legend Label: Enter a label for the legend to display in the viewer.

Chart Data Dialog

When you first open the Chart Data dialog, you can select a **Dataset name** to associate with the chart. The list is populated with all of the datasets in the report's dataset collection.

This dialog also gives you access to the following related pages.

General Page

Name: Enter a name for the chart that is unique within the report. This name is displayed in the Document Outline and in XML exports.

Tooltip: A textual label for the report item used to include TITLE or ALT attributes in HTML reports.

Dataset Name: Select a dataset to associate with the chart. The combo box is populated with all of the datasets in the report's dataset collection.

Series Values Page

Add at least one Value series to determine the size of the chart element. Click the plus sign button to enable the General tab. Once you have one or more value series in place, you can use the arrow buttons to change the order or the X button to delete them.

Another way to add Chart Series Values is to drag fields from the Report Explorer onto the tray along the top edge of the chart that reads **Drop data fields here**.

If you have already added values, you can right-click any value displayed in the UI along the top of the chart and choose **Edit** to open this dialog.

The Series Values page has the following tabs.

General

The General tab of the Series Values page allows you to control different items depending on the Chart Type you have chosen.

For all Chart types

Series label: Enter an expression to use as a series label to display in the legend.

For Scatter or Bubble Chart types

X: Enter an expression to use as an X value.

Y: Enter an expression to use as a Y value.

Size: If the chart type is bubble, enter an expression to use as the bubble size value.

For Stock Chart

High: Enter an expression to use as the high value.

Low: Enter an expression to use as the low value.

Open: Enter an expression to use as the open value.

Close: Enter an expression to use as the close value.

For Column, Bar, Line, Pie, Area, Doughnut, Funnel, Pyramid, ThreeLineBreak, Kagi, Renko, PointAndFigure, or DotPlot Chart types

Value: Enter an expression to use as a series value.

For Column, Line, or Area Chart types

Chart Type: For Composite charts, select the chart type to use in combination with other chart types within the same plot area. The available chart types are:

- Column Plain
- Column Stacked
- Column Percent Stacked
- Area Plain
- Area Stacked
- Area Percent Stacked
- Line Plain
- Smooth Line

Y-Axis: Select the Y-axis from the list of available Y-axes.

Styles

Line/Border

These properties control the appearance of the border of bars or columns, or the lines, depending on the type of chart.

Style: Choose one of the enumerated styles for the lines.

Width: Choose a width value between 0.25pt and 20pt for the thickness of the lines.

Color: Choose a Web or Custom color to use for the lines.

Background Fill Color

These properties control the appearance of the background of the series values.

Fill Color: Choose a Web or Custom color to fill the background.

Gradient: Choose from one of the following gradient styles.

- **None**: No gradient is used. A single color (defined by the **Fill Color** property above) is used to fill the area and the **Gradient End Color** property remains disabled.
- **LeftRight**: A gradient is used. The **Fill Color** property defines the color at the left, and the **Gradient End Color** property defines the color at the right. The two colors are gradually blended in between these areas.
- **TopBottom**: A gradient is used. The **Fill Color** property defines the color at the top, and the **Gradient End Color** property defines the color at the bottom. The two colors are gradually blended in between these areas.
- **Center**: A gradient is used. The **Fill Color** property defines the color at the center, and the **Gradient End Color** property defines the color at the edges. The two colors are gradually blended in between these areas.
- **DiagonalLeft**: A gradient is used. The **Fill Color** property defines the color at the top left, and the **Gradient End Color** property defines the color at the bottom right. The two colors are gradually blended in between these areas.
- **DiagonalRight**: A gradient is used. The **Fill Color** property defines the color at the top right, and the **Gradient End Color** property defines the color at the bottom left. The two colors are gradually blended in between these areas.
- **HorizontalCenter**: A gradient is used. The **Gradient End Color** property defines the horizontal band of color across the center, and the **Fill Color** property defines the color at the top and bottom. The two colors are gradually blended in between these areas.
- **VerticalCenter**: A gradient is used. The **Gradient End Color** property defines the vertical band of color across the center, and the **Fill Color** property defines the color at the left and right. The two colors are gradually blended in between these areas.

Gradient End Color: When you choose any gradient style other than **None**, this property becomes available. Choose a Web or Custom color to blend with the **Fill Color** in the background of the series.

Markers

Marker type: Choose one of the following values to determine the shape of the marker or whether one is displayed.

- **None** - Markers are not used. (Default)
- **Square** - Markers are square.
- **Circle** - Markers are circular.
- **Diamond** - Markers are diamond shaped.
- **Triangle** - Markers are triangular.
- **Cross** - Markers are cross shaped.
- **Auto** - A shape is chosen automatically.

Marker size: Enter a value between **2pt** and **10pt** to determine the size of the plotting area of the markers.

Plot data as secondary: If the chart type is Column, Bar, or DotPlot, you can select this check box and select whether to use a Line or Points to show the data.

Labels

Show point labels: Select this check box to display a label for each chart value. Selecting this box enables the disabled properties on this page.

Data label: Enter a value to use as the label, or select **<Expression...>** to open the Expression Editor.

Format code: Select one of the provided format codes or use a custom .NET formatting code to format dates or numbers. For more information, see MSDN's Formatting Types topic.

Position: Leave **Auto** selected to use the default point label position for the chart type, or select an enumerated value to position the labels.

Angle: Enter the value in tenths of degrees to use for the angle of the point label text. The default (0°) position denotes no angle and renders regular horizontal text.

Font

Family: Choose the font family name.

Size: Choose the size in points for the font.

Style: Choose **Normal** or **Italic**.

Weight: Choose from Lighter, Thin, ExtraLight, Light, Normal, Medium, SemiBold, Bold, ExtraBold, Heavy, and Bolder.

Color: Select a Web or custom color for the font.

Decoration: Choose from None, Underline, Overline, and LineThrough.

Action

Choose from the following actions to perform when the user clicks on the chart element.

None: The default behavior is to do nothing when a user clicks the chart element at run time.

Jump to report: For drill-through reporting, select this option and provide the name of a local report, the relative path of a report in another folder, or the full path of a report on another server.

Parameters

- **Name**: Supply the exact names of any parameters required for the targeted report. Note that parameter names you supply in this must match parameters in the target report.

  > **Important:** The Parameter Name must exactly match the name of the parameter in the detail report. If any parameter is spelled differently, capitalized differently, or if an expected parameter is not supplied, the drill-through report will fail.

- **Value**: Enter a Parameter Value to pass to the detail report. This value must evaluate to a valid value for the parameter.
- **Omit**: Select this check box to omit this parameter from the report.

Jump to bookmark: Select this option and provide a valid Bookmark ID to allow the user to jump to the report control with that Bookmark ID.

Jump to URL: Select this option and provide a valid URL to create a hyperlink to a Web page.

Data Output

Element name: Enter a name to be used in the XML output for this chart element.

Output: Choose Yes or No to decide whether to include this chart element in the XML output.

Category Groups Page

Add Category Groups to group data and provide labels for the chart elements. Click the **Add** button to enable the General tab. Once you have one or more category groups in place, you can use the arrow buttons to change the order or the X button to delete them.

Another way to add Category Groups is to drag fields from the Report Explorer onto the tray along the bottom edge of the chart that reads **Drop category fields here**.

If you have already added values, you can right-click the value displayed in the UI along the bottom of the chart and choose **Edit** to open this dialog.

The Category Groups page has the following tabs.

General

Name: Enter a name for the group that is unique within the report. This name can be called in code.

Group on: Enter an expression to use for grouping the data.

Label: Enter an expression to use as a label for the group. You can select **<Expression...>** to open the Expression Editor.

Parent group: For use in recursive hierarchies. Enter an expression to use as the parent group.

Filters

You need to provide three values to add a new filter to the collection: Expression, Operator, and Value.

Expression: Enter the expression to use for evaluating whether data should be included in the group.

Operator: Select from the following operators to decide how to compare the expression to the left with the value to the right:

- **Equal** Only choose data for which the value on the left is equal to the value on the right.
- **Like** Only choose data for which the value on the left is similar to the value on the right.
  For more information on using the **Like** operator, see the MSDN Web site.
- **NotEqual** Only choose data for which the value on the left is not equal to the value on the right.
- **GreaterThan** Only choose data for which the value on the left is greater than the value on the right.
- **GreaterThanOrEqual** Only choose data for which the value on the left is greater than or equal to the value on the right.
- **LessThan** Only choose data for which the value on the left is less than the value on the right.
- **LessThanOrEqual** Only choose data for which the value on the left is less than or equal to the value on the right.
- **TopN** Only choose items from the value on the left which are the top number specified in the value on the right.
- **BottomN** Only choose items from the value on the left which are the bottom number specified in the value on the right.
- **TopPercent** Only choose items from the value on the left which are the top percent specified in the value on the right.
- **BottomPercent** Only choose items from the value on the left which are the bottom percent specified in the value on the right.
- **In** Only choose items from the value on the left which are in the array of values specified on the right.
  Selecting this operator enables the Values list at the bottom.
- **Between** Only choose items from the value on the left which fall between the pair of values you specify on the right. Selecting this operator enables two Value boxes instead of one.

Value: Enter a value to compare with the expression on the left based on the selected operator. For multiple values used with the **Between** operator, the lower two value boxes are enabled.

Values: When you choose the **In** operator, you can enter as many values as you need in this list.

Sorting

The Sorting tab of Category Groups page allows you to enter new sort expressions and remove or change the order of them using the X or arrow buttons. For each sort expression in this list, you can also choose the direction.

Expression: Enter an expression by which to sort the data in the group.

Direction: Select whether you want to sort the data in an Ascending or Descending direction.

Data Output

Element name: Enter a name to be used in the XML output for this group.

Collection: Enter a name to be used in the XML output for the collection of all instances of this group.

Output: Choose Yes or No to decide whether to include this group in the XML output.

Series Groups Page

Optionally add Series Groups for extra levels of data (for example, Orders by Country can be broken down by year as well). Labels for the series are displayed in the chart legend. Click the **Add** button to open the General page. Once you have one or more series groups in place, you can use the arrow buttons to change the order or the X button to delete them.

Another way to add Series Groups is to drag fields from the Report Explorer onto the tray along the right edge of the chart that reads **Optionally drop series fields here**.

If you have already added values, you can right-click the value displayed in the UI along the right edge of the chart and choose **Edit** to open this dialog.

The Series Groups page has the following tabs.

General

Name: Enter a name for the group that is unique within the report. This name can be called in code.

Group on: Enter an expression to use for grouping the data.

Label: Enter an expression to use as a label for the group. You can select **<Expression...>** to open the Expression Editor.

Parent group: For use in recursive hierarchies. Enter an expression to use as the parent group.

Filters

You need to provide three values to add a new filter to the collection: Expression, Operator, and Value.

Expression: Enter the expression to use for evaluating whether data should be included in the group.

Operator: Select from the following operators to decide how to compare the expression to the left with the value to the right:

- **Equal** Only choose data for which the value on the left is equal to the value on the right.
- **Like** Only choose data for which the value on the left is similar to the value on the right.
  For more information on using the **Like** operator, see the MSDN Web site.
- **NotEqual** Only choose data for which the value on the left is not equal to the value on the right.
- **GreaterThan** Only choose data for which the value on the left is greater than the value on the right.
- **GreaterThanOrEqual** Only choose data for which the value on the left is greater than or equal to the value on the right.
- **LessThan** Only choose data for which the value on the left is less than the value on the right.
- **LessThanOrEqual** Only choose data for which the value on the left is less than or equal to the value on the right.
- **TopN** Only choose items from the value on the left which are the top number specified in the value on the right.

- **BottomN** Only choose items from the value on the left which are the bottom number specified in the value on the right.
- **TopPercent** Only choose items from the value on the left which are the top percent specified in the value on the right.
- **BottomPercent** Only choose items from the value on the left which are the bottom percent specified in the value on the right.
- **In** Only choose items from the value on the left which are in the array of values specified on the right. Selecting this operator enables the Values list at the bottom.
- **Between** Only choose items from the value on the left which fall between the pair of values you specify on the right. Selecting this operator enables two Value boxes instead of one.

Value: Enter a value to compare with the expression on the left based on the selected operator. For multiple values used with the **Between** operator, the lower two value boxes are enabled.

Values: When you choose the **In** operator, you can enter as many values as you need in this list.

Sorting

The Sorting tab of Series Groups page allows you to enter new sort expressions and remove or change the order of them using the X or arrow buttons. For each sort expression in this list, you can also choose the direction.

Expression: Enter an expression by which to sort the data in the group.

Direction: Select whether you want to sort the data in an Ascending or Descending direction.

Data Output

Element name: Enter a name to be used in the XML output for this group.

Collection: Enter a name to be used in the XML output for the collection of all instances of this group.

Output: Choose Yes or No to decide whether to include this group in the XML output.

Filters Page

Chart Data Filters Page

You need to provide three values to add a new filter to the collection: Expression, Operator, and Value.

Expression: Enter the expression to use for evaluating whether data should be included in the chart.

Operator: Select from the following operators to decide how to compare the expression to the left with the value to the right:

- **Equal** Only choose data for which the value on the left is equal to the value on the right.
- **Like** Only choose data for which the value on the left is similar to the value on the right. For more information on using the **Like** operator, see the MSDN Web site.
- **NotEqual** Only choose data for which the value on the left is not equal to the value on the right.
- **GreaterThan** Only choose data for which the value on the left is greater than the value on the right.
- **GreaterThanOrEqual** Only choose data for which the value on the left is greater than or equal to the value on the right.
- **LessThan** Only choose data for which the value on the left is less than the value on the right.
- **LessThanOrEqual** Only choose data for which the value on the left is less than or equal to the value on the right.
- **TopN** Only choose items from the value on the left which are the top number specified in the value on the right.
- **BottomN** Only choose items from the value on the left which are the bottom number specified in the value on the right.

- **TopPercent** Only choose items from the value on the left which are the top percent specified in the value on the right.
- **BottomPercent** Only choose items from the value on the left which are the bottom percent specified in the value on the right.
- **In** Only choose items from the value on the left which are in the array of values specified on the right. Selecting this operator enables the Values list at the bottom.
- **Between** Only choose items from the value on the left which fall between the pair of values you specify on the right. Selecting this operator enables two Value boxes instead of one.

Value: Enter a value to compare with the expression on the left based on the selected operator. For multiple values used with the **Between** operator, the lower two value boxes are enabled.

Values: When you choose the **In** operator, you can enter as many values as you need in this list.

Value: Enter a value to compare with the expression on the left based on the selected operator. For multiple values (used with the **In** and **Between** operators) separate values using commas.

Data Output Page

Chart Data Output Page

Element name: Enter a name to be used in the XML output for the chart.

Output: Choose one between Auto, Yes, No or Contents Only to decide whether to include this group in the XML output.

CheckBox (Page Report)

In ActiveReports, you can use the CheckBox control to represent a Boolean value in a report. By default, it appears as a small box with text to the right. If the value evaluates to True, the small box appears with a check mark; if False, the box is empty. By default, the checkbox is empty.

Checkbox Dialog

Properties for the CheckBox are available in the Checkbox dialog. To open it, with the CheckBox control selected on the report, under the Properties Window, click the **Property dialog** link.

The Checkbox dialog lets you set properties on the report control with the following pages.

> **Note**: You can select the **<Expression...>** option in many of these properties to create an expression to determine the value. For properties with enumerated values, the values are listed under **Constants** in the **Fields** tree view on the left side of the Expression Editor. You can also access the **Expression Editor** from the context menu of the CheckBox control.

General

Name: Enter a name for the checkbox that is unique within the report. This name is displayed in the Document Outline and in XML exports. You can only use underscore (_) as a special character in the Name field. Other special characters such as period (.), space ( ), forward slash (/), back slash (\), exclamation (!), and hyphen (-) are not supported.

Tooltip: A textual label for the report item used to include TITLE or ALT attributes in HTML reports.

Value: Enter an expression or a static label, or choose a field expression from the drop-down list. You can access the expression editor by selecting **<Expression...>** in the list. The value of this expression or text is displayed in the report to

the right of the checkbox.

Visibility

Initial visibility

- **Visible** - The checkbox is visible when the report runs.
- **Hidden** - The checkbox is hidden when the report runs.
- **Expression** - Use an expression with a Boolean result to decide whether the checkbox is visible. For example, on a "Free Shipping" checkbox, you could use the expression to see whether the ShippingCountry is international. A value of true hides the checkbox, false shows it.

Visibility can be toggled by another report control: Select this checkbox to specify a report control to use as a toggle to show or hide the checkbox. Then specify the TextBox control to display with a toggle image button. When the user clicks the TextBox control, the checkbox changes between visible and hidden.

Appearance

Border

Style: Select a style for the border.

Width: Enter a value in points to set the width of the border.

Color: Select a color to use for the border, or select the **<Expression...>** option to open the Expression Editor and create an expression that evaluates to a .NET color.

Background

Color: Select a color to use for the background of the checkbox.

Image: Enter an image to use for the background of the checkbox.

Font

Family: Select a font family name or a theme font.

Size: Choose the size in points for the font or use a theme.

Style: Choose **Normal** or **Italic** or select a theme.

Weight: Choose an enumerated weight value or select a theme.

Color: Choose a color to use for the text.

Decoration: Choose from **None**, **Underline**, **Overline**, or **LineThrough**.

Format

Format code: Select one of the common numeric formats provided or use a custom .NET formatting code to format dates or numbers. For more information, see MSDN's Formatting Types topic.

Line Spacing: This property sets the space between lines of text.

Line height: This property sets the height of each line of text.

> **Note**: This property only affects HTML output.

Character Spacing: This property sets the space between characters of text.

Text direction and writing mode

Direction: Choose **LTR** for left to right, or **RTL** for right to left.

Mode: Choose **lr-tb** for left right top bottom (normal horizontal text) or **tb-rl** for top bottom right left (vertical text on its side).

Alignment

Alignment

Vertical alignment: Choose **Top**, **Middle**, **Bottom**, or the **<Expression...>** option.

Horizontal alignment: Choose **General**, **Left**, **Center**, **Right**, **Justify**, or the **<Expression...>** option.

Justification method: Choose **Auto**, **Distribute**, **DistributAllLines**, or the **<Expression...>** option.

Wrap mode: Choose **NoWrap**, **WordWrap**, or **CharWrap**.

> **Note:** You must select **Justify** in the **Horizontal alignment** property to enable the **Justification method** property options.

Amount of space to leave around report control

- **Top padding**: Set the top padding in points.
- **Left padding**: Set the left padding in points.
- **Right padding**: Set the right padding in points.
- **Bottom padding**: Set the bottom padding in points.

Data Output

Element Name: Enter a name to be used in the XML output for this checkbox.

Output: Choose **Auto**, **Yes**, or **No** to decide whether to include this checkbox in the XML output. Auto exports the contents of the checkbox only when the value is not a constant.

Render as: Choose **Auto**, **Element**, or **Attribute** to decide whether to render checkboxes as Attributes or Elements in the exported XML file. Auto uses the report's setting for this property.

Attribute example: <table1 checkbox3="Report created on: 7/26/2005 1:13:00 PM">

Element example: <table1> <checkbox3>Report created on: 7/26/2005 1:13:28 PM</checkbox3>

Container

The **Container** report control is a container for other items. There are a number of ways in which you can use it to enhance your reports.

Visual Groupings

You can place report controls within the Container to group them visually, and to make it easier at design time to move a group of report controls.

> **Note:** Drawing a container around existing items does not contain them. Instead you must drag the items into the container.

You can use a container as a border for your report pages, and set border properties to create purely visual effects within your report, and even display an image behind a group of report controls by setting the **BackgroundImage** property of the Container.

Anchoring Items

Probably the best usage of the Container report control is to anchor report controls which may otherwise be pushed down by a vertically expanding data region. For example, if you have a group of textboxes below a table with some of them to the left or right, any of them directly below the table are pushed down below the expanded table at run time, while the upper textboxes remain where you placed them at design time. To prevent this from happening, place the group of textboxes within a container.

Container Dialog

Properties for the Container are available in the Container dialog. To open it, with the Container control selected on the report, under the Properties Window, click the **Property dialog** link.

The Container dialog lets you set properties on the report control with the following pages.

> **Note**: You can select the **<Expression...>** option in many of these properties to create an expression to determine the value. For properties with enumerated values, the values are listed under **Constants** in the **Fields** tree view on the left side of the Expression Editor.

General

Name: Enter a name for the container that is unique within the report. This name can be called in code. You can only use underscore (_) as a special character in the Name field. Other special characters such as period (.), space ( ), forward slash (/), back slash (\), exclamation (!), and hyphen (-) are not supported.

Page breaks:

- **Insert a page break before this container**: Insert a page break before the container.
- **Insert a page break after this container**: Insert a page break after the container.

Appearance

Background

Color: Select a color to use for the background of the container.

Image: Specify the background image of container using Expression or Data Visualizer, or directly open the image file on your system.

> **Note**: For Page/RDL reports, if the Hatch and Gradient background styles are set using Data Visualizers, these are not displayed at design time.

Border

Style: Select a style for the border.

Width: Enter a value in points to set the width of the border.

Color: Select a color to use for the border, or select the **<Expression...>** option to open the Expression Editor and create an expression that evaluates to a .NET color.

Rounded Rectangle: Specify the radius for each corner of the shape independently. Drag the handlers available at each corner of the shape to set the value of the radius at each corner.

> **Note:** To enable specific corners, check the CheckBox available near each corner of the Container control.

Visibility

Initial visibility

- **Visible**: The container is visible when the report runs.
- **Hidden**: The container is hidden when the report runs.
- **Expression**: Use an expression with a Boolean result to decide whether the container is visible. True for hidden, False for visible.

Visibility can be toggled by another report control: Select this check box to display a toggle image next to another report control. This enables the drop-down box below where you can specify the TextBox control that toggles the visibility of the container. The user can click the toggle item to show or hide this container.

Navigation

Document map label: Enter an expression to use as a label to represent this item in the table of contents (document map).

Bookmark ID: Enter an expression to use as a locator for this container. You will then be able to provide a bookmark link to this item from another report control using a **Jump to bookmark** action.

Data Output

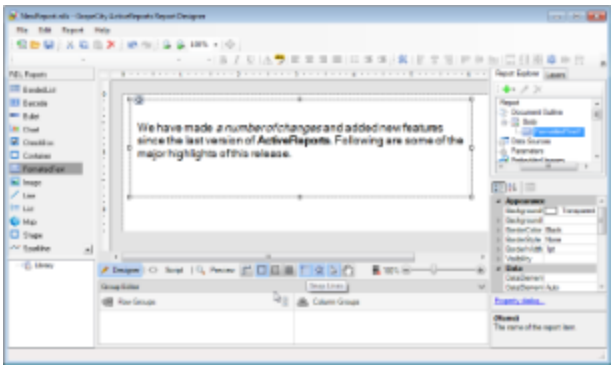
The Data Output page of the Container dialog allows you to control the following properties when you export to XML:

- **Element name**: Enter a name to be used in the XML output for this container.
- **Output**: Choose **Auto**, **Yes**, **No**, or **Contents only** to decide whether to include the contents of this container in the XML output. Choosing **Auto** exports the contents of the container only when the value is not a constant.

Formatted Text

The FormattedText report control can perform mail merge operations, plus it displays richly formatted text in HTML. To format text in the FormattedText report control, enter HTML code into the **Html** property.

The design time editor displays the HTML text with the applied formatting, so you can view the text just as it will be displayed at run time in the **Designer** tab.



Supported HTML Tags

If you use valid HTML tags that are not in this list, ActiveReports ignores them.

⚠ **Important**: All text used in the **Html** property must be enclosed in the **<body></body>** tags.

Tag	Description
<%MergeFieldName%>	**Inserts a mail merge field.**
<!-- -- >	Defines a comment
<!DOCTYPE>	Defines the document type
<a>	Defines an anchor
<abbr>	Defines an abbreviation
<acronym>	Defines an acronym
<address>	Defines an address element
<b>	Defines bold text
<base />	Defines a base URL for all the links in a page
<bdo>	Defines the direction of text display
<big>	Defines big text
<blockquote>	Defines a long quotation
<body>	**Defines the body element (Required)**
 	Inserts a single line break
<caption>	Defines a table caption
	Defines centered text
<cite>	Defines a citation
<code>	Defines computer code text
<col>	Defines attributes for table columns
<dd>	Defines a definition description

<del>	Defines deleted text
<dir>	Defines a directory list
	Defines a section in a document
<dfn>	Defines a definition term
<dl>	Defines a definition list
<dt>	Defines a definition term
	Defines emphasized text
<h1> to <h6>	Defines header 1 to header 6
<head>	Defines information about the document
<hr />	Defines a horizontal rule
<html>	Defines an html document
<i>	Defines italic text

<img />	Defines an image. Use **src** attribute to define path of the image, for example:
	`<img src="https://.../images/image.png"/>`
	To refer an image embedded within the report, specify the image name:
	`<img src="image_name"/>`

<ins>	Defines inserted text
<kbd>	Defines keyboard text
	Defines a list item
<link>	Defines a link
<map>	Defines an image map
<menu>	Defines a menu list
	Defines an ordered list
<p>	Defines a paragraph
<pre>	Defines preformatted text
<q>	Defines a short quotation
<s>	Defines strikethrough text
<samp>	Defines sample computer code
<small>	Defines small text
	Defines a section in a document
<strike>	Defines strikethrough text
	Defines strong text
<style>	Defines a style definition

<sub>	Defines subscripted text
<sup>	Defines superscripted text
<table>	Defines a table
<tbody>	Defines a table body
<td>	Defines a table cell
<tfoot>	Defines a table footer
<th>	Defines a table header
<thead>	Defines a table header
<tr>	Defines a table row
<tt>	Defines teletype text
<u>	Defines underlined text
	Defines an unordered list

⚠ **Caution**: To enter **&** in the HTML property, you need to use **&**.

Formatted Text Dialog

Properties for the FormattedText report control are available in the Formatted Text dialog. To open it, with the control selected on the report, under the Properties Window, click the **Property dialog** link.

The Formatted Text dialog lets you set properties on the report control with the following page.

General

Name: Enter a name for the FormattedText that is unique within the report. This name can be called in code. You can only use underscore (_) as a special character in the Name field. Other special characters such as period (.), space ( ), forward slash (/), back slash (\), exclamation (!), and hyphen (-) are not supported.

Tooltip: A textual label for the report item used to include TITLE or ALT attributes in HTML reports.

Visibility

Initial visibility

- **Visible**: The FormattedText is visible when the report runs.
- **Hidden**: The FormattedText is hidden when the report runs.
- **Expression**: Use an expression with a Boolean result to decide whether the FormattedText is visible. True for hidden, false for visible.

Visibility can be toggled by another report control: Select this check box to display a toggle image next to another report control. This enables the drop-down box below where you can specify the TextBox control that toggles the visibility of the FormattedText. The user can click the toggle item to show or hide this FormattedText.

Navigation

Document map label: Enter an expression to use as a label to represent this item in the table of contents (document

map).

Bookmark ID: Enter an expression to use as a locator for this FormattedText. You will then be able to provide a bookmark link to this item from another report control using a **Jump to bookmark** action.

Appearance

Background

Color: Select a color to use for the background of the FormattedText.

Image: Enter an image to use for the background of the FormattedText.

Border

Style: Select a style for the border.

Width: Enter a value in points to set the width of the border.

Color: Select a color to use for the border, or select the **<Expression...>** option to open the Expression Editor and create an expression that evaluates to a .NET color.

Data Output

Element name: Enter a name to be used in the XML output for this FormattedText report control.

Output: Choose **Auto**, **Yes**, **No, Contents Only** to decide whether to include this FormattedText in the XML output. Choosing **Auto** exports the contents of the FormattedText report control.

Mail Merge

Click the plus sign button to add a new mail merge field to the FormattedText, and delete them using the X button. Once you add one or more fields, you can reorder them using the arrow buttons.

Field: Enter a name for the field that is unique within the report. This is used in the Html property inside <%FieldName%> tags to display the field in the formatted text.

Value: Enter an expression to pull data into the control for mail merge operations.

Here is a very simple example of HTML code that you can use to add mail merge fields to formatted text. This example assumes that you have added two mail merge fields named **Field1** and **Field2**.

Paste this code in the Html property of the FormattedText control.

```
<body><p>This is <%Field1/%> and this is <%Field2/%>.</p></body>
```

Image

The **Image** report control displays an image that you embed in the report, add to the project, store in a database, or access through a URL. You can choose the **Image Source** in the Properties Window after you place the Image report control on the report.

Embedded Images

The benefit of using an embedded image is that there is no separate image file to locate or to keep track of when you move the report between projects. The drawback is that the larger the file size of the image you embed, the more inflated

your report file size becomes.

To embed an image in your report

1. In the **Report** menu, select **Embedded Images**.
2. Click under the **Image** column to reveal an ellipsis button (**...**) and select an image file from your local files. The **Name** and **MimeType** columns are filled in automatically and the image file's data is stored in the report definition.
3. Click to select the Image report control, and in the Properties grid, set the **Source** property to **Embedded**.
4. Drop down the **Value** property and select the image from the list of embedded images.

Data Visualizer Images

You can use a data visualizer to display data in small graphs that are easy to comprehend.

To add a data visualizer image to your report

1. Click to select the Image report control, and in the Properties grid, drop down the **Value** property and select **<Data Visualizer...>**.
2. In the Data Visualizers dialog that appears, select the Visualizer Type that you want to use, Icon Set, Range Bar, or Data Bar.
3. Use expressions related to your data to set the rest of the values in the dialog.

Project Images

You may have an image that you want to use in multiple reports, for example a logo. In such cases, you can store it as a project image. This not only allows you to quickly locate the correct image for new reports in the project, but also makes it easier when you update your logo, as you will not need to search through every report to replace embedded images. Another benefit is that the images are distributed with your application.

To store an image in your Visual Studio project

1. Right-click the project in the Solution Explorer and select **Add**, then **Add Existing Item** and navigate to the image.
2. Click to select the Image report control, and in the Properties grid, set the **Source** property to **External**.
3. Drop down the **Value** property and select the image from the list of project images.

Database Images

Product catalogues are probably the most common scenario in which images stored in a database are used in reports. Place the Image report control in a data region to use database images that repeat for every row of data.

Keep in mind that you cannot use database images in Page Headers and Page Footers because these sections cannot have value expressions that refer to fields.

To use a database image in an Image report control

1. Click to select the Image report control, and in the Properties grid, set the **Source** property to **Database**.
2. Drop down the **Value** property and select the field containing the image.

> **Note**: Microsoft Access database images are generally stored as OLE objects which the Image report control cannot read.

Web Images

You can also use any image to which you can navigate via a URL. The benefit of this is that images stored in this way add nothing to the file size of the project or of the report, but the drawback is that if the web based image is moved, it will no longer show up in your report.

To use a Web image

1. Click to select the Image report control, and in the Properties grid, set the **Source** property to **External**.
2. In the **Value** property, enter the URL for the image.

Image Dialog

Properties for the Image are available in the Image dialog. To open it, with the Image control selected on the report, under the Properties Window, click the **Property dialog** link.

The Image dialog lets you set properties on the report control with the following pages.

> **Note**: You can select **<Expression...>** within many of these properties to open the Expression Editor.

General

Name: Enter a name for the image that is unique within the report. This name can be called in code. You can only use underscore (_) as a special character in the Name field. Other special characters such as period (.), space ( ), forward slash (/), back slash (\), exclamation (!), and hyphen (-) are not supported.

Tooltip: A textual label for the report item used to include TITLE or ALT attributes in HTML reports.

Image Value: Enter the name of the image to display. Depending on the Image Source chosen below, you can give a path to the image, select an image to embed, or pull images from a database. This property also allows you to choose the **<Data Visualizer...>** option to launch a dialog that will let you build a data visualization expression.

Image Source: Select whether the image comes from a source that is **External**, **Embedded**, or **Database**.

MIME Type: Select the MIME type of the image chosen.

Visibility

Initial visibility

- **Visible**: The image is visible when the report runs.
- **Hidden**: The image is hidden when the report runs.
- **Expression**: Use an expression with a Boolean result to decide whether the image is visible. True for hidden, False for visible.

Visibility can be toggled by another report control: Select this check box to display a toggle image next to another report control. This enables the drop-down box below where you can specify the TextBox control that toggles the visibility of the image. The user can click the toggle item to show or hide this image.

Navigation

Action

Select one of the following actions to perform when a user clicks on this image.

None: The default behavior is to do nothing when a user clicks the image at run time.

Jump to report: For drill-through reporting, select this option and provide the name of a local report, the relative

path of a report in another folder, or the full path of a report on another server.

Parameters: Supply parameters to the targeted report by entering the **Name** of each parameter, the **Value** to send to the targeted report, or whether to **Omit** the parameter. Note that parameter names you supply must exactly match parameters in the target report.

> **Tip**: You can remove or change the order of parameters using the X and arrow buttons.

Jump to bookmark: Select this option and provide a valid Bookmark ID to allow the user to jump to the report control with that Bookmark ID.

Jump to URL: Select this option and provide a valid URL to create a hyperlink to a Web page.

Document map label: Enter an expression to use as a label to represent this item in the table of contents (document map).

Bookmark ID: Enter an expression to use as a locator for this image. You will then be able to provide a bookmark link to this item from another report control using a **Jump to bookmark** action.

InputField

The **InputField** report control provides support for editable fields in an exported PDF report where the InputField's value can be modified.

> **Note:** The InputField control is part of the Professional Edition. With the Standard license, the InputField control is not displayed in an exported file.

You can choose one of the two report control types – Text and Checkbox, in the **InputType** property. Each selected type has its own set of properties.

In case of the Text type, the InputField control gets the set of properties of the TextBox control. If the Checkbox type is selected, then the control inherits the set of properties of the CheckBox control.

Limitations

- (Preview limitation) Border drawing is displayed inside the control area.
- (Preview limitation) A checkmark inside the InputField CheckBox is shown only if the field size allows as a checkmark can't be displayed partly.
- (Preview limitation) Some properties are not supported at preview, like Multiline (False), Password, Required.
- With the PDF export filter, the InputField control is exported as a static, non-editable control.
- (PDF limitation) Limited set of enum values for the BorderStyle property - None, Solid, Dashed, Inset.
- (PDF limitation) Padding, VerticalAlign, Justify settings are not available.
- (PDF limitation) Properties related to the Text/ Font settings are not available for the InputType control as Checkbox.
- (PDF limitation) Border properties can only be set for all sides at once (the option for each side is not available).

Line

The **Line** report control can be used to visually separate data regions in a report layout. You can set properties in the Properties grid or the Line Dialog to control the appearance of the line, and to control when the line is rendered.

Important Properties

Property	Description
LineColor	Gets or sets the color of the line.
LineStyle	Gets or sets the pen style used to draw the line. The line styles include Transparent, Solid, Dash, Dot, DashDot, DashDotDot, or Double.
LineWidth	Gets or sets the pen width of the line in points.
Visibility	Indicates whether the line is visible or hidden.
LayerName	Gets or sets the name of the containing layer.
EndPoint	Set the vertical and horizontal end points of the line.
Location	Gets or sets the position of the top-left corner of the report item in relation to its container.

Line Dialog

Properties for the Line are available in the Line dialog. To open it, with the Line control selected on the report, under the Properties Window, click the **Property dialog** link.

The Line dialog lets you set properties on the report control with the following pages.

General

Name: Enter a name for the line that is unique within the report. This name can be called in code. You can only use underscore (_) as a special character in the Name field. Other special characters such as period (.), space ( ), forward slash (/), back slash (\), exclamation (!), and hyphen (-) are not supported.

Visibility

The Visibility page of the Line dialog allows you to control the following items:

Initial visibility

- **Visible**: The line is visible when the report runs.
- **Hidden**: The line is hidden when the report runs.
- **Expression**: Use an expression with a Boolean result to decide whether the line is visible. True for hidden, False for visible.

Visibility can be toggled by another report control: Select this check box to display a toggle image next to another report control. This enables the drop-down box below where you can specify the TextBox control that toggles the visibility of the line. The user can click the toggle item to show or hide this line.

Navigation

Document map label: Enter an expression to use as a label to represent this item in the table of contents (document map).

Bookmark ID: Enter an expression to use as a locator for this line. You will then be able to provide a bookmark link to this item from another report control using a **Jump to bookmark** action.

List

The **List** data region repeats any report controls it contains for every record in the dataset. Unlike other data regions, the **List** is free-form, so you can arrange report controls in any configuration you like.

Grouping in the list is done on the details group.

List Dialog

Properties for the List are available in the List dialog. To open it, with the List control selected on the report, under the Properties Window, click the **Property dialog** link.

The List dialog lets you set properties on the report control with the following pages.

> **Note**: You can select **<Expression...>** within these properties to create an expression to determine the value.

General

Name: Enter a name for the list that is unique within the report. This name can be called in code. You can only use underscore (_) as a special character in the Name field. Other special characters such as period (.), space ( ), forward slash (/), back slash (\), exclamation (!), and hyphen (-) are not supported.

Tooltip: A textual label for the report item used to include TITLE or ALT attributes in HTML reports.

Dataset name: Select a dataset to associate with the list. The drop-down list is populated with all of the datasets in the report's dataset collection.

Has own page numbering: Select to indicate whether this List is in its own section with regards to pagination.

Page Breaks: Select any of the following options to apply to each instance of the list.

- Insert a page break before this list
- Insert a page break after this list
- Fit list on a single page if possible

Detail Grouping

Detail grouping is useful when you do not want to repeat values within the details. When a detail grouping is set, the value repeats for each distinct result of the grouping expression instead of for each row of data. For example, if you use the Customers table of the NorthWind database to create a list of countries without setting the details grouping, each country is listed as many times as there are customers in that country. If you set the details grouping to `=Fields!Country.Value` each country is listed only once.

> **Note**: If the detail grouping expression you use results in a value that is distinct for every row of data, a customer number for example, you will see no difference in the results.

The Detail Grouping page of the List dialog has the following tabs.

General

Name: Enter a name for the group that is unique within the report. This property cannot be set until after a **Group on** expression is supplied. You can only use underscore (_) as a special character in the Name field. Other special characters such as period (.), space ( ), forward slash (/), back slash (\), exclamation (!), and hyphen (-) are not supported.

Group on: Enter an expression to use for grouping the data.

Document map label: Enter an expression to use as a label to represent this item in the table of contents (document map).

Parent group: For use in recursive hierarchies. Enter an expression to use as the parent group.

Filters

You need to provide three values to add a new filter to the collection: Expression, Operator, and Value.

Expression: Enter the expression to use for evaluating whether data should be included in the group.

Operator: Select from the following operators to decide how to compare the expression to the left with the value to the right.

- **Equal** Only choose data for which the value on the left is equal to the value on the right.
- **Like** Only choose data for which the value on the left is similar to the value on the right.
  For more information on using the **Like** operator, see the MSDN Web site.
- **NotEqual** Only choose data for which the value on the left is not equal to the value on the right.
- **GreaterThan** Only choose data for which the value on the left is greater than the value on the right.
- **GreaterThanOrEqual** Only choose data for which the value on the left is greater than or equal to the value on the right.
- **LessThan** Only choose data for which the value on the left is less than the value on the right.
- **LessThanOrEqual** Only choose data for which the value on the left is less than or equal to the value on the right.
- **TopN** Only choose items from the value on the left which are the top number specified in the value on the right.
- **BottomN** Only choose items from the value on the left which are the bottom number specified in the value on the right.
- **TopPercent** Only choose items from the value on the left which are the top percent specified in the value on the right.
- **BottomPercent** Only choose items from the value on the left which are the bottom percent specified in the value on the right.
- **In** Only choose items from the value on the left which are in the array of values specified on the right.
  Selecting this operator enables the Values list at the bottom.
- **Between** Only choose items from the value on the left which fall between the pair of values you specify on the right. Selecting this operator enables two Value boxes instead of one.

Value: Enter a value to compare with the expression on the left based on the selected operator. For multiple values used with the **Between** operator, the lower two value boxes are enabled.

Values: When you choose the **In** operator, you can enter as many values as you need in this list.

Sorting

Click the plus sign button to enter new sort expressions, and remove them using the X button.

In the **Expression** box, enter an expression by which to sort the data in the group, and under **Direction**, select **Ascending** or **Descending** for the selected sort expression.

Data Output

Element name: Enter a name to be used in the XML output for this group.

Collection: Enter a name to be used in the XML output for the collection of all instances of this group.

Output: Choose **Yes** or **No** to decide whether to include this group in the XML output.

Layout

Page break at start: Inserts a page break before the group.

Page break at end: Inserts a page break after the group.

Has own page numbering: Used in conjunction with the "Page Number in Section" and "Total Pages in Section" properties, tells the report that the group constitutes a new page numbering section.

Visibility

Initial visibility

- **Visible**: The list is visible when the report runs.
- **Hidden**: The list is hidden when the report runs.
- **Expression**: Use an expression with a Boolean result to decide whether the list is visible. True for hidden, false for visible.

Visibility can be toggled by another report control: Select this check box to display a toggle image next to another report control. This enables the drop-down box below where you can specify the TextBox control that toggles the visibility of the list. The user can click the toggle item to show or hide this list.

Navigation

Document map label: Enter an expression to use as a label to represent this item in the table of contents (document map).

Bookmark ID: Enter an expression to use as a locator for this list. You will then be able to provide a bookmark link to this item from another report control using a **Jump to bookmark** action.

Filters

You need to provide three values to add a new filter to the collection: Expression, Operator, and Value.

Expression: Enter the expression to use for evaluating whether data should be included in the group.

Operator: Select from the following operators to decide how to compare the expression to the left with the value to the right.

- **Equal** Only choose data for which the value on the left is equal to the value on the right.
- **Like** Only choose data for which the value on the left is similar to the value on the right.
  For more information on using the **Like** operator, see the MSDN Web site.
- **NotEqual** Only choose data for which the value on the left is not equal to the value on the right.
- **GreaterThan** Only choose data for which the value on the left is greater than the value on the right.
- **GreaterThanOrEqual** Only choose data for which the value on the left is greater than or equal to the value on the right.
- **LessThan** Only choose data for which the value on the left is less than the value on the right.
- **LessThanOrEqual** Only choose data for which the value on the left is less than or equal to the value on the right.
- **TopN** Only choose items from the value on the left which are the top number specified in the value on the right.
- **BottomN** Only choose items from the value on the left which are the bottom number specified in the value on the right.
- **TopPercent** Only choose items from the value on the left which are the top percent specified in the value on the right.
- **BottomPercent** Only choose items from the value on the left which are the bottom percent specified in the value on the right.
- **In** Only choose items from the value on the left which are in the array of values specified on the right.
  Selecting this operator enables the Values list at the bottom.

- **Between** Only choose items from the value on the left which fall between the pair of values you specify on the right. Selecting this operator enables two Value boxes instead of one.

Value: Enter a value to compare with the expression on the left based on the selected operator. For multiple values used with the **Between** operator, the lower two value boxes are enabled.

Values: When you choose the **In** operator, you can enter as many values as you need in this list.

Sorting

Click the plus sign button to enter new sort expressions, and remove them using the X button.

In the **Expression** box, enter an expression by which to sort the data in the group, and under **Direction**, select **Ascending** or **Descending** for the selected sort expression.

Data Output

Element name: Enter a name to be used in the XML output for this list.

Output: Choose **Auto**, **Yes**, or **No** to decide whether to include this List in the XML output. Choosing **Auto** exports the contents of the list.

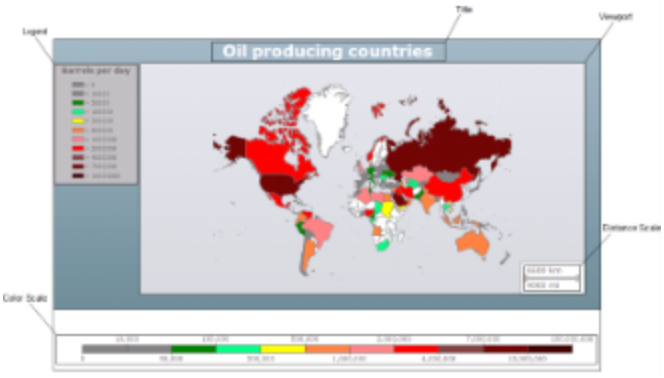
Instance element name: Enter a name to be used in the XML output for the data element for instances of the list. This name is ignored if you have specified a detail grouping.

Instance element output: Choose Yes or No to decide whether to include the instances of the list in the XML output. This is ignored if you have specified a detail grouping.

Map

The Map data region is a professional edition feature that shows your business data against a geographical background. You can create different types of map, depending on the type of information you want to communicate in your report.

The Map data region consists of the following basic elements:



Title

Map Title describes the theme or subject of the map. The purpose of map title is to tell the viewer of what he is looking at. You can add multiple titles to the Map using the **MapTitleDesigner Collection Editor**.

For more information, see Create a Map.

Viewport

Viewport refers to the area on the map where data is displayed against a geographical background. It specifies the coordinates, projection system, parallels and meridians, center point, and scale of the map. In other words, it is a map element that actually displays geographical data and occupies most area of the map control depending on the location and dock position of other map elements. For more information, see Create a Map.

The **Map Viewport** dialog lets you set properties with the following pages.

General

Coordinate system: Specify the coordinate system of the viewport. Select from Planar, Geographic, or select the <**Expression...**> option to open the Expression Editor and create an expression.

Projection: Specify the projection of the map. Tile layers must use the Mercator projection.

Minimum X: Specify the minimum X coordinate of the map in degrees.

Maximum X: Specify the maximum X coordinate of the map in degrees.

Minimum Y: Specify the minimum Y coordinate of the map in degrees.

Maximum Y: Specify the maximum Y coordinate of the map in degrees.

Projection Center X: Specify the X coordinate of the projection center in degrees.

Projection Center Y: Specify the Y coordinate of the projection center in degrees.

Minimum zoom: Specify the minimum zoom value.

Maximum zoom: Specify the maximum zoom value.

Map resolution: Enables the viewport to simplify vector data for polygon and line layers.

Show grid lines below the map: Specify whether to show the grid lines above or below the content of the map.

Meridians

Hide meridians: Specify whether to hide meridians.

Interval: Specify the spacing between the grid lines in degrees.

Line

- **Style:** Choose from None, Dotted, Dashed, Solid, Double, Groove, Ridge, Inset, WindowInset, Outset, or select the <**Expression...**> option to open the Expression Editor and create an expression.
- **Width:** Specify the width of the line.
- **Color:** Select a Web or custom color for the line.

Show labels: Specify whether to show labels for meridians on the map.

Format: Specify the format string to display numeric labels.

Position: Specify the position of the meridians on the map.

Font

- **Family:** Choose the font family name.
- **Size:** Choose the size in points for the font.
- **Style:** Choose Normal or Italic.
- **Weight:** Choose from Lighter, Thin, ExtraLight, Light, Normal, Medium, SemiBold, Bold, ExtraBold, Heavy, and Bolder.
- **Color:** Select a Web or custom color for the font.
- **Decoration:** Choose from None, Underline, Overline, LineThrough, or select the <**Expression...**> option to

open the Expression Editor and create an expression.

Parallels

Hide parallels: Specify whether to hide parallels.

Interval: Specify the spacing between the grid lines in degrees.

Line

- **Style:** Choose from None, Dotted, Dashed, Solid, Double, Groove, Ridge, Inset, WindowInset, Outset, or select the <**Expression...**> option to open the Expression Editor and create an expression.
- **Width:** Specify the width of the line.
- **Color:** Select a Web or custom color for the line.

Show labels: Specify whether to show labels for parallels on the map.

Format: Specify the format string to display numeric labels.

Position: Specify the position of the parallels on the map.

Font

- **Family:** Choose the font family name.
- **Size:** Choose the size in points for the font.
- **Style:** Choose Normal or Italic.
- **Weight:** Choose from Lighter, Thin, ExtraLight, Light, Normal, Medium, SemiBold, Bold, ExtraBold, Heavy, and Bolder.
- **Color:** Select a Web or custom color for the font.
- **Decoration:** Choose from None, Underline, Overline, LineThrough, or select the <**Expression...**> option to open the Expression Editor and create an expression.

View

Center and zoom: Specify how the map viewport zooms and centers during the report processing.

- Custom
- Center map to show a map element
- Center map to show a map layer
- Center map to show all map elements

View Center X: Specify the X coordinate of the current view center.

View Center Y: Specify the Y coordinate of the current view center.

Zoom level: Specify the zoom level of the map view.

Appearance

Border

- **Style:** Choose an enumerated style for the border.
- **Width:** Set a width value in points between 0.25pt and 20pt.
- **Color:** Select a Web or custom color for the border.

Background

- **Color:** Select a color to use for the background of the Viewport.

- **Gradient:** Select a color to use for the border, or select the <**Expression...**> option to open the Expression Editor and create an expression that evaluates to a .NET color.
- **Gradient End Color:** Select a color to use for the end color of the background gradient.
- **Pattern:** Select the hatching pattern of a report control.

Shadow offset: Specify the size of the shadow. Shadow offsets are drawn to the right and below an element.

Legend

A legend on a map provides valuable information to users for interpreting the map data visualization rules such as color, size, and marker type differences for map elements on a layer. By default, a single Legend item already exists in the legends collection which can be used by all layers to display items. You can also create additional legends to use them individually with layers that have associated rules to display items in the legend.

Legends are added in the **LegendDesigner Collection Editor**. For more information, see Create a Map.

Distance Scale

A distance scale helps a user to understand the scale of the map. Distance on a map is not the same as the actual real-world distance, so a distance scale shows that a certain distance on the map equals a certain distance in a real-world. In distance scale, the distance is displayed in both miles and kilometers. The scale range and values are automatically calculated using the viewport boundaries, projection type, and zoom level. For more information, see Create a Map.

The Map Distance Scale dialog lets you set properties with the following pages.

> **Note**: You can select **<Expression...>** within these properties to create an expression to determine the value.

General

Location

- **Position:** Specify the docking position of the distance scale panel. Choose from TopCenter, TopLeft, TopRight, LeftTop, LeftCenter, LeftBottom, RightTop, RightCenter, RightBottom, BottomRight, or select the <**Expression...**> option to open the Expression Editor and create an expression.
- **Show distance scale outside the viewport:** Specify whether the panel is docked inside or outside of the map viewport.

Scale

- **Color:** Select the fill color for the distance scale bar.
- **Border color:** Select the border color for the distance scale bar.

Appearance

Border

- **Style:** Choose from None, Dotted, Dashed, Solid, Double, Groove, Ridge, Inset, WindowInset, or Outset.
- **Width:** Choose the width of the border line.
- **Color:** Select a color for the border.

Background

- **Color:** Select a color to use for the background of the distance scale.
- **Gradient:** Select a color to use for the border, or select the <**Expression...**> option to open the Expression Editor and create an expression that evaluates to a .NET color.

- **Gradient End Color:** Select a color to use for the end color of the background gradient.
- **Pattern:** Select the hatching pattern of the distance scale panel from the list of patterns, or select the <**Expression...**> option to open the Expression Editor and create an expression.

Shadow offset: Specify the size of the shadow of the distance scale panel. Shadow offsets are drawn to the right and below an element.

Font

Family: Choose the font family name.

Size: Choose the size in points for the font.

Style: Choose Normal or Italic, or select the <**Expression...**> option to open the Expression Editor and create an expression.

Weight: Choose from Lighter, Thin, ExtraLight, Light, Normal, Medium, SemiBold, Bold, ExtraBold, Heavy, and Bolder.

Color: Select a Web or custom color for the font.

Decoration: Choose from None, Underline, Overline and LineThrough, or select the <**Expression...**> option to open the Expression Editor and create an expression.

Visibility

Initial visibility

- **Visible**: The distance scale is visible when the report runs.
- **Hidden**: The distance scale is hidden when the report runs.
- **Expression**: Use an expression with a Boolean result to decide whether the distance scale is visible. True for hidden, false for visible.

Navigation

Action

Select one of the following actions to perform when a user clicks on the distance scale element.

- **None:** The default behavior is to do nothing when a user clicks the distance scale element at run time.
- **Jump to URL:** Select this option and provide a valid URL to create a hyperlink to a Web page.
- **Jump to bookmark:** Select this option and provide a valid Bookmark ID to allow the user to jump to the report control with that Bookmark ID.
- **Jump to report:** For drill-through reporting, select this option and provide the name of a local report, the relative path of a report in another folder, or the full path of a report on another server.

Color Scale

A color scale helps a user to understand the range of colors that are used for data visualization on a layer. A map has just one color scale and multiple layers can provide data for it. For more information, see Create a Map.

The Map Color Scale dialog lets you set properties with the following pages.

> **Note**: You can select **<Expression...>** within these properties to create an expression to determine the value.

General

Location

- **Position:** Specify the docking position of the color scale panel. Choose from TopCenter, TopLeft, TopRight, LeftTop, LeftCenter, LeftBottom, RightTop, RightCenter, RightBottom, BottomRight, or select the <**Expression...**> option to open the Expression Editor and create an expression.
- **Show color scale outside the viewport:** Specify whether the panel is docked inside or outside of the map viewport.

Color bar

- **Border color:** Specify the outline color for color scale divisions.
- **Range gap color:** Specify color to fill color divisions for undefined range values.

Labels

Display: Specify whether to display color scale labels on the color scale panel. Select from Auto, ShowMiddleValue, ShowBorderValue, or select the <**Expression...**> option to open the Expression Editor and create an expression.

Hide end labels: Specify whether to display first and last labels on the color scale panel.

Format: Specify the format string to display numeric labels.

Placement: Speciy the position of the color scale labels on the color scale panel. Select from Alternate, Top, Bottom, or select the <**Expression...**> option to open the Expression Editor and create an expression.

Interval: Specify the frequency of the labels on the color scale panel. A value of 0 means no labels are displayed.

Tick mark length: Specify the length of the tick marks on the color scale panel.

Title

Text: Specify the text of the color scale panel.

Font

- **Family:** Choose the font family name.
- **Size:** Choose the size in points for the font.
- **Style:** Choose Normal, Italic or select the <**Expression...**> option to open the Expression Editor and create an expression.
- **Weight:** Choose from Lighter, Thin, ExtraLight, Light, Normal, Medium, SemiBold, Bold, ExtraBold, Heavy, and Bolder.
- **Color:** Select a Web or custom color for the font.
- **Decoration:** Choose from None, Underline, Overline, LineThrough, or select the <**Expression...**> option to open the Expression Editor and create an expression.

Appearance

Border

- **Style:** Choose from None, Dotted, Dashed, Solid, Double, Groove, Ridge, Inset, WindowInset, or Outset.
- **Width:** Specify the width of the border.
- **Color:** Specify a color for the border.

Background

- **Color:** Select a color to use for the background of the distance scale.
- **Gradient:** Specify whether and how to use color gradients in the color scale background. Select from None, LeftRight, TopBottom, Center, DiagonalLeft, DiagonalRight, HorizontalCenter, VerticalCenter, or select the <**Expression...**> option to open the Expression Editor and create an expression.

- **Gradient End Color:** Select a color to use for the end color of the background gradient.
- **Pattern:** Select the hatching pattern of the color scale panel from the list of patterns, or select the <**Expression...**> option to open the Expression Editor and create an expression.

Shadow offset: Specify the size of the shadow of the color scale panel. Shadow offsets are drawn to the right and below an element.

Font

Family: Choose the font family name.

Size: Choose the size in points for the font.

Style: Choose Normal, Italic or select the <**Expression...**> option to open the Expression Editor and create an expression.

Weight: Choose from Lighter, Thin, ExtraLight, Light, Normal, Medium, SemiBold, Bold, ExtraBold, Heavy, and Bolder.

Color: Select a Web or custom color for the font.

Decoration: Choose from None, Underline, Overline, LineThrough, or select the <**Expression...**> option to open the Expression Editor and create an expression.

Visibility

Initial visibility

- **Visible**: The color scale is visible when the report runs.
- **Hidden**: The color scale is hidden when the report runs.
- **Expression**: Use an expression with a Boolean result to decide whether the color scale is visible. True for hidden, false for visible.

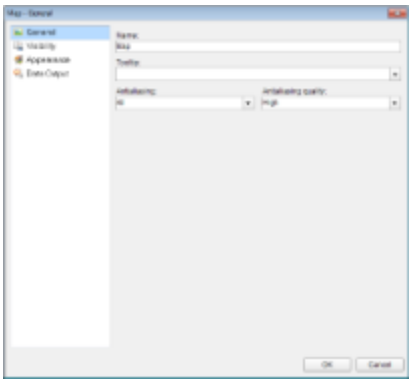
Navigation

Action

Select one of the following actions to perform when a user clicks on the color scale element.

- **None:** The default behavior is to do nothing when a user clicks the color scale element at run time.
- **Jump to URL:** Select this option and provide a valid URL to create a hyperlink to a Web page.
- **Jump to bookmark:** Select this option and provide a valid Bookmark ID to allow the user to jump to the report control with that Bookmark ID.
- **Jump to report:** For drill-through reporting, select this option and provide the name of a local report, the relative path of a report in another folder, or the full path of a report on another server.

Map Dialog



Properties for the Map are available in the Map dialog. To open it, with the Map control selected on the report, under the Properties Window, click the **Property dialog** link.

The Map dialog lets you set properties on the report control with the following pages.

Note: You can select **<Expression...>** within these properties to create an expression to determine the value.

General

Name: Enter a name for the map that is unique within the report. This name is called in code. You can only use underscore (_) as a special character in the Name field. Other special characters such as period (.), space ( ), forward slash (/), back slash (\), exclamation (!), and hyphen (-) are not supported.

Tooltip: A textual label for the report item used to include TITLE or ALT attributes in HTML reports.

Antialiasing: Select the smoothing mode to apply to all map control elements. Choose All, None, Text, Graphic, or select the <**Expression...**> option to open the Expression Editor and create an expression.

Antialiasing quality: Select the quality for antialiasing. Choose High, Normal, SystemDefault, or select the <**Expression...**> option to open the Expression Editor and create an expression.

Visibility

Initial visibility

- **Visible**: The map is visible when the report runs.
- **Hidden**: The map is hidden when the report runs.
- **Expression**: Use an expression with a Boolean result to decide whether the map is visible. True for hidden, false for visible.

Visibility can be toggled by another report control: Select this check box to display a toggle image next to another report control. This enables the drop-down box below where you can specify the TextBox control that toggles the visibility of the map. The user can click the toggle item to show or hide this map.

Appearance

Border

- **Style:** Choose from None, Dotted, Dashed, Solid, Double, Groove, Ridge, Inset, WindowInset, or Outset.
- **Width:** Specify the width of the border.
- **Color:** Select a Web or custom color for the font.

Background

- **Color:** Select a color to use for the background of the map.
- **Gradient:** Specify whether and how to use color gradients in the color scale background. Select from None, LeftRight, TopBottom, Center, DiagonalLeft, DiagonalRight, HorizontalCenter, VerticalCenter, or select the <**Expression...**> option to open the Expression Editor and create an expression.
- **Gradient End Color:** Select a color to use for the end color of the background gradient.
- **Pattern:** Select the hatching pattern of a report control.

Data Output

Element name: Enter a name to be used in the XML output for this map.

Output: Choose **Auto**, **Yes**, or **No** to decide whether to include this map in the XML output. Choosing **Auto** exports the contents of the map.

Layers

A map is a collection of layers that display data on the map control.

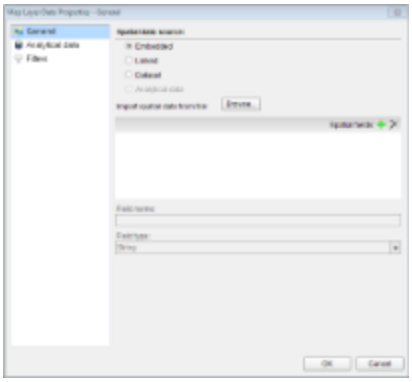
- **Polygon layer:** Display outlines of areas or markers for the polygon center point. See, Use a Polygon Layer for more information.
- **Point layer:** Display markers for point locations. See, Use a Point Layer for more information.
- **Line layer:** Display lines for paths or routes. See, Use a Line Layer for more information.
- **Tile layer:** Adds a Bing map tiles background. See, Use a Tile Layer for more information.

A map can have one or more layers. You can load these layers on top of each other to create a more detailed map. For example, a polygon layer can represent the borders of a country, a line can represent transportation routes, a point can represent the locations and a tile can add a virtual earth background on the map. See, Use Layers for more information.

Map layer element appearance:

- Properties that you set on a polygon layer, line layer and a point layer apply to all map elements on that layer, whether or not the map elements are embedded in the report definition.
- Properties that you set for rules apply to all map elements on a layer. All data visualization options apply only to map elements that are associated with spatial data.

Map Layer Data Dialog



The Map layer Data dialog is used to set up spatial and analytical data for the map control. For more information on spatial and analytical data, see Add Data.

To access Map Layer Data Properties dialog

1. Click the map until the map panes appears.
2. Right click the layers pane and select **Add <*layerName*> Layer**. This adds a new layer to the map and opens the **Map Layer Data Properties** dialog

Or, in case you already have a layer added to the map control, then follow these steps:

1. Click the map until the map panes appears.
2. In the layer pane, right-click the existing layer and select **Layer Data** to open **Map Layer Data Properties** dialog.

General

Spatial data source: Select one of the spatial data source connection types:

- **Embedded:** The map layer data is loaded from the .shp data source that you embed into the map layer by indicating the .shp file in the **Import spatial data from file:** field. This field appears below when you select this option.

  Spatial fields: Use the plus sign button to add a field, and the X button to delete a field. For each newly added spatial field, you must specify the name and type in the corresponding fields below.

  Field name: Enter a name of a spatial field.

  Field type: Select the type of a spatial field from the list.

- **Linked:** The map layer data is linked to the .shp file and is uploaded at report rendering. You select this type of data source by indicating the .shp file in the **File Name** field that appears.
- **Dataset:** The map layer data is loaded from the data source of the report. In the **Dataset** field and in the **Field name** field that appear below, select a dataset from the bound data source and a dataset field.

  > ⚠ **Caution**: In **Field name**, simply type the name of the dataset field that contains spatial data. For example, enter the dataset field name as ***StateName***, not as ***=[StateName]***.

- **Analytical data:** The map layer data is loaded from the the analytical dataset of the bound report data source. In the **Field name** field that appears below, you must set the name of the data field that contains spatial data in the Analytical dataset.

  > ⚠ **Caution**: In **Field name**, enter the data field name as ***=[StateName]***, not as ***StateName***.

Analytical Data

Dataset: Select the dataset for the analytical data to be displayed on the map layer.

Match: Use the plus sign button to add a relationship between a spatial data field and an analytical data field.

Spatial field: A field with spatial data that specifies an element on the map surface, for example, boundaries of a country.

Analytical field: A field with analytical data that displays information on the related map element, for example, the country population.

Filters

The **Filters** page of the **Map Layer Data Properties** dialog allows you to filter the data that is included in the map. Use

the plus sign button to add a filter, and the arrow and X buttons to move or delete filters. You need to provide three values to add a new filter to the collection.

Expression: Enter the expression to use for evaluating whether data should be included in the map.

Operator: Select from the following operators to decide how to compare the expression to the left with the value to the right:

- **Equal** Only choose data for which the value on the left is equal to the value on the right.
- **Like** Only choose data for which the value on the left is similar to the value on the right. For more information on using the **Like** operator, see the MSDN Web site.
- **NotEqual** Only choose data for which the value on the left is not equal to the value on the right.
- **GreaterThan** Only choose data for which the value on the left is greater than the value on the right.
- **GreaterThanOrEqual** Only choose data for which the value on the left is greater than or equal to the value on the right.
- **LessThan** Only choose data for which the value on the left is less than the value on the right.
- **LessThanOrEqual** Only choose data for which the value on the left is less than or equal to the value on the right.
- **TopN** Only choose items from the value on the left which are the top number specified in the value on the right.
- **BottomN** Only choose items from the value on the left which are the bottom number specified in the value on the right.
- **TopPercent** Only choose items from the value on the left which are the top percent specified in the value on the right.
- **BottomPercent** Only choose items from the value on the left which are the bottom percent specified in the value on the right.
- **In** Only choose items from the value on the left which are in the array of values specified on the right. Selecting this operator enables the Values list at the bottom.
- **Between** Only choose items from the value on the left which fall between the pair of values you specify on the right. Selecting this operator enables two Value boxes instead of one.

Value: Enter a value to compare with the expression on the left based on the selected operator. For multiple values used with the **Between** operator, the lower two value boxes are enabled.

Values: When you choose the **In** operator, you can enter as many values as you need in this list.

Value: Enter a value to compare with the expression on the left based on the selected operator. For multiple values (used with the **In** and **Between** operators) separate values using commas.

Overflow Place Holder

In a Page report, the OverflowPlaceHolder control is a rectangular placeholder for data that does not fit inside the fixed size of a **List**, **BandedList**, **Tablix** or **Table** data region. When you link a data region to an OverflowPlaceHolder, this control gets its **Size** property values from the **FixedSize** of the data region it is linked with.

You can also place multiple OverflowPlaceHolder controls in a report to create different looks for your data output. Link a data region to an OverflowPlaceHolder control and then link that OverflowPlaceHolder control to another OverflowPlaceHolder control. Two common layouts that you can create through this process are:

- **Multiple Page Layout:** Place the data region on the first page of the report and OverflowPlaceHolder controls on subsequent pages to create a layout with overflow data on multiple pages.

> **Note:** If a page contains only an OverflowPlaceHolder with no data to display, the empty page does not render. However, if the page also contains any control with static data, the page renders. To skip rendering the page, set **GrapeCity.ActiveReports.PageReportModel.Page.ThrowIfPlaceHoldersEmpty** property to True.

- **Columnar Report Layout:** Place the data region and the OverflowPlaceHolder on the same page of the report to create a layout that displays data in a columnar format like the one in the following image.

Data overflow to an OverflowPlaceHolder

You can bind overflow data from a data region to an OverflowPlaceHolder control or from an OverflowPlaceHolder control to another OverflowPlaceHolder control in a report. The following steps take you through the process:

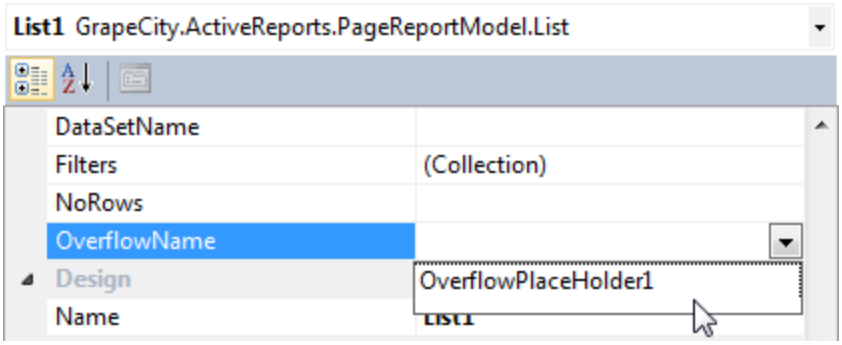
These steps assume that you have already added a Page Report template to your project, connected it to a data source and added a DataSet. See Connect to a Data Source and Add a Dataset for more information.

To link a data region to an OverflowPlaceHolder control

When your data goes beyond the fixed size of a data region, you can create a link from the data region to enable flow of data into the OverflowPlaceHolder.

1. From the Visual Studio toolbox, on the Page1 tab of the report, drag and drop a data region like List onto the design surface and set its FixedSize property.
2. If the data goes beyond the fixed size of the data region, from the Visual Studio toolbox, on Page2 tab of the report, drag and drop an OverflowPlaceHolder control (OverflowPlaceHolder1 by default) onto the design surface.
3. On the Page1 design surface, select the data region placed above and go to the Properties Window.
4. In the Properties Window, go to the **GrapeCity.ActiveReports.PageReportModel.DataRegion.OverflowName** property and from the dropdown list, select the name of the OverflowPlaceHolder control you added earlier.

   The following image shows the Properties Window of a List data region (List1) where OverflowPlaceHolder1 is set in the OverflowName property.



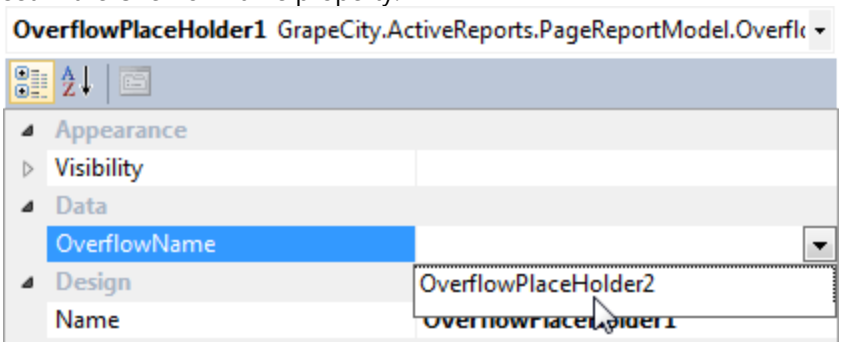
> **Tip:** Depending on your layout requirements, you can place the OverflowPlaceHolder control on the same page tab as the data region or a different page tab.

To link an OverflowPlaceHolder control to another OverflowPlaceHolder control

You can place additional OverflowPlaceHolder controls, to display data that flows beyond the first OverflowPlaceHolder control.

1. From the Visual Studio toolbox, drag and drop another OverflowPlaceHolder control like OverflowPlaceHolder2 onto the design surface.
2. In the Designer, select the OverflowPlaceHolder1 control that contains overflow data and go to the properties window.
3. In the Properties Window, go to **OverflowName** property and select the name of the new OverflowPlaceHolder control you placed above. For e.g., in OverflowPlaceHolder1, set the OverflowName property to OverflowPlaceHolder2.

   The following image shows the Properties Window of OverflowPlaceHolder1 where the OverflowPlaceHolder2 is set in the OverflowName property.



> **Caution:** In a report with multiple OverflowPlaceHolder controls, link the OverflowPlaceHolder controls to their respective data regions and other OverflowPlaceHolder controls such that the overflow chain does not break.

Shape

The Shape report control is used to display one of the available shape types on a report. You can add a shape report control to a report by dragging it from the toolbox and dropping it onto the report design surface.

In the **ShapeStyle** property of the Shape report control, you can select **Rectangle**, **RoundRect** or **Ellipse**, or you can use an expression to assign fields, datasets, parameters, constants, operations or common values to it. You can highlight different sections or parts of a report using a shape report control. For example, you can use a Rectangle as border around different report controls or the entire page or you can use an Ellipse to highlight a note on your report.

Shape Dialog

Properties for the Shape are available in the Shape dialog. To open it, with the Shape control selected on the report, under the Properties Window, click the **Property dialog** link.

The Shape dialog lets you set properties on the report control with the following pages.

> **Note**: You can select **<Expression...>** within many of these properties to open the Expression Editor.

General

Name: Enter a name for the image that is unique within the report. This name can be called in code. You can only use underscore (_) as a special character in the Name field. Other special characters such as period (.), space ( ), forward slash (/), back slash (\), exclamation (!), and hyphen (-) are not supported.

Shape Style: Choose **Rectangle**, **RoundRect** or **Ellipse** from the dropdown list.

Rounded Rectangle: When the Shape type is set to **RoundRect**, you can specify the radius for each corner of the shape independently. Drag the handlers ◆ available at each corner of the shape to set the value of the radius at each corner.

> **Note:** To enable specific corners, check the CheckBox available near each corner of the Shape control.

Appearance

Background

Color: Select a color to use for the background of the Shape.

Image: Specify the background image of shape using Expression or Data Visualizer, or directly open the image file on your system.

> **Note**: For Page/RDL reports, if the Hatch and Gradient background styles are set using Data Visualizers, these are not displayed at design time.

Border

Style: Select a style for the border.

Width: Enter a value in points to set the width of the border.

Color: Select a color to use for the border, or select the **<Expression...>** option to open the Expression Editor and create an expression that evaluates to a .NET color.

Visibility

Initial visibility

- **Visible**: The shape is visible when the report runs.
- **Hidden**: The shape is hidden when the report runs.
- **Expression**: Use an expression with a Boolean result to decide whether the shape is visible. True for hidden, false for visible.

Visibility can be toggled by another report control: Select this check box to display a toggle shape next to another report control. This enables the drop-down box where you can specify the TextBox control which, if clicked, toggles the visibility of the shape.

Navigation

Document map label: Enter an expression to use as a label to represent this item in the table of contents (document map).

Bookmark ID: Enter an expression to use as a locator for this shape. You will then be able to provide a bookmark link to this item from another report control using a **Jump to bookmark** action.

Data Output

Element name: Enter a name to be used in the XML output for this shape report control.

Output: Choose **Auto**, **Yes**, **No, Contents Only** to decide whether to include this Shape in the XML output. Choosing **Auto** exports the contents of the Shape report control.

Sparkline

You can use the Sparkline report control as a simple means of displaying the trend of data in a small graph. The Sparkline displays the most recent value as the rightmost data point and compares it with earlier values on a scale, allowing you to view general changes in data over time. With the height similar to the surrounding text and the width not more than 14 letters wide, the sparkline fits well in dashboards, reports, and other documents.

Customize the Sparkline control with the following types.

Type	Description
Line	The Line sparkline is widely used in financial and economic data analysis and is based on a continuous flow of data. The currency exchange rates, changes in price are the examples of application of this type of sparklines.
Columns	The Column sparkline is used for sports scores, cash register receipts, and other cases where previous values and the current value do not closely influence one another. In this case you are dealing with discrete data points, and not a continuous flow of data as in the Line sparkline.
Whiskers	The Whisker sparkline is typically used in win/loss/tie or true/false scenarios. This type is similar to the Column sparkline, but it renders a tie (0 value) in a different manner. The bars in a whisker sparkline render below the baseline for a negative value, above the baseline for a positive value, and on the baseline for a zero value.
Area	The Area sparkline is similar to the Line sparkline but visually you see the space under the line as shaded.
StackedBar	The Stacked Bar sparkline is presented as a horizontal bar with different segment lengths marked by distinct color hues. The Stacked bar illustrates how the various segments of a part-to-whole relationship correspond to one another - the largest segment represents the highest value and the change in brightness

	indicates a new value on a scale.

Sparkline Dialog

Properties for the Sparkline are available in the Sparkline dialog. To open it, with the Sparkline control selected on the report, under the Properties Window, click the **Property dialog** link.

The Sparkline dialog lets you set properties on the report control with the following pages.

General

Name: Enter a name for the sparkline that is unique within the report. This name can be called in code. You can only use underscore (_) as a special character in the Name field. Other special characters such as period (.), space ( ), forward slash (/), back slash (\), exclamation (!), and hyphen (-) are not supported.

Data

Value: Enter an expression to use as the sparkline value.

Name: Enter a name for the sparkline that is unique within the report. This name can be called in code. A name is created automatically if you do not enter one.

Group on: Enter an expression to use for grouping the data. If you open the expression editor, you can select a field from the dataset.

Detail Grouping: Enter an expression to use if you do not want to repeat values within the details. If you open the expression editor, you can select a field from the dataset.

Parent Group: For use in recursive hierarchies. Enter an expression to use as the parent group.

Appearance

Sparkline Type: Choose **Line**, **Columns**, **Whiskers**, **Area** or **StackedBar**. Each of these types has its own set of Appearance properties that appears when you select the type.

Line Type Appearance Properties

Last point marker is visible: Select to display a marker at the last point on the sparkline.

Marker Color: Select a color to use for the last point marker, or select the <**Expression...**> option to open the Expression Editor and create an expression that evaluates to a .NET color.

Line Style

Color: Select a color to use for the line, or select the <**Expression...**> option to open the Expression Editor and create an expression that evaluates to a .NET color.

Width: Enter a value in points to set the width of the line.

Enable Wall Range: Select this check box to display a wall range for the sparkline. Selecting this box enables the rest of the properties in this section.

Lower Bound: Select a value or enter an expression that defines the lower bound of the wall range.

Upper Bound: Select a value or enter an expression that defines the upper bound of the wall range.

Wall Range Backdrop

Fill Color: Select a color to use for the wall range, or select the <**Expression...**> option to open the Expression Editor and create an expression that evaluates to a .NET color.

Gradient: Choose the type of gradient to use for the backdrop: **None**, **LeftRight**, **TopBottom**, **Center**, **DiagonalLeft**, **DiagonalRight**, **HorizontalCenter** or **VerticalCenter**.

Gradient End Color: Select a color to use for the end of the wall range gradient, or select the <**Expression...**> option to open the Expression Editor and create an expression that evaluates to a .NET color.

Columns Type Appearance Properties

Fill Color: Select a color to use for the fill of the sparkline, or select the <**Expression...**> option to open the Expression Editor and create an expression that evaluates to a .NET color.

Maximum Column Width: Select the maximum width of columns in the sparkline. If blank, all columns are sized to fit.

Enable Wall Range: Select this check box to display a wall range for the sparkline. Selecting this box enables the rest of the properties in this section.

Lower Bound: Select a value or enter an expression that defines the lower bound of the wall range.

Upper Bound: Select a value or enter an expression that defines the upper bound of the wall range.

Wall Range Backdrop

Fill Color: Select a color to use for the wall range, or select the <**Expression...**> option to open the Expression Editor and create an expression that evaluates to a .NET color.

Gradient: Choose the type of gradient from these choices: **None**, **LeftRight**, **TopBottom**, **Center**, **DiagonalLeft**, **DiagonalRight**, **HorizontalCenter** or **VerticalCenter**.

Gradient End Color: Select a color to use for the end of the wall range gradient, or select the <**Expression...**> option to open the Expression Editor and create an expression that evaluates to a .NET color.

Whiskers Type Appearance Properties

Fill Color: Select a color to use for the fill of the sparkline, or select the <**Expression...**> option to open the Expression Editor and create an expression that evaluates to a .NET color.

Maximum Column Width: Select the maximum width of columns in the sparkline. If blank, all columns are sized to fit.

Enable Wall Range: Select this check box to display a wall range for the sparkline. Selecting this box enables the rest of the properties in this section.

Lower Bound: Select a value or enter an expression that defines the lower bound of the wall range.

Upper Bound: Select a value or enter an expression that defines the upper bound of the wall range.

Wall Range Backdrop

Fill Color: Select a color to use for the wall range, or select the <**Expression...**> option to open the Expression Editor and create an expression that evaluates to a .NET color.

Gradient: Choose the type of gradient from these choices: **None**, **LeftRight**, **TopBottom**, **Center**, **DiagonalLeft**, **DiagonalRight**, **HorizontalCenter** or **VerticalCenter**.

Gradient End Color: Select a color to use for the end of the wall range gradient, or select the <**Expression...**> option to open the Expression Editor and create an expression that evaluates to a .NET color.

Area Type Appearance Properties

Fill Color: Select a color to use for the fill of the sparkline, or select the <**Expression...**> option to open the Expression Editor and create an expression that evaluates to a .NET color.

Enable Wall Range: Select this check box to display a wall range for the sparkline. Selecting this box enables the rest of the properties in this section.

Lower Bound: Select a value or enter an expression that defines the lower bound of the wall range.

Upper Bound: Select a value or enter an expression that defines the upper bound of the wall range.

Wall Range Backdrop

Fill Color: Select a color to use for the wall range, or select the <**Expression...**> option to open the Expression Editor and create an expression that evaluates to a .NET color.

Gradient: Choose the type of gradient from these choices: **None**, **LeftRight**, **TopBottom**, **Center**, **DiagonalLeft**, **DiagonalRight**, **HorizontalCenter** or **VerticalCenter**.

Gradient End Color: Select a color to use for the end of the wall range gradient, or select the <**Expression...**> option to open the Expression Editor and create an expression that evaluates to a .NET color.

StackedBar Type Appearance Properties

Fill Color: Select a color to use for the base color of the stacked bars, or select the <**Expression...**> option to open the Expression Editor and create an expression that evaluates to a .NET color. The other colors of the stacked bars are calculated using this based color.

Visibility

Initial visibility

- **Visible:** The sparkline is visible when the report runs.
- **Hidden:** The sparkline is hidden when the report runs.
- **Expression:** Use an expression with a Boolean result to decide whether the sparkline is visible. True for hidden, False for visible.

Visibility can be toggled by another report control: Select this check box to display a toggle image next to another report item. This enables the drop-down box below where you can specify the TextBox control which, if clicked, toggles the visibility of the sparkline in the Viewer.

Navigation

Document map label: Enter an expression to use as a label to represent this item in the table of contents (document map).

Bookmark ID: Enter an expression to use as a locator for this sparkline. You will then be able to provide a bookmark link to this item from another report item using a **Jump to bookmark** action.

Filters

You need to provide three values to add a new filter to the collection: Expression, Operator, and Value.

Expression: Enter the expression to use for evaluating whether data should be included in the group.

Operator: Select from the following operators to decide how to compare the expression to the left with the value to the right.

- **Equal** Only choose data for which the value on the left is equal to the value on the right.
- **Like** Only choose data for which the value on the left is similar to the value on the right. For more information on using the **Like** operator, see the MSDN Web site.
- **NotEqual** Only choose data for which the value on the left is not equal to the value on the right.
- **GreaterThan** Only choose data for which the value on the left is greater than the value on the right.
- **GreaterThanOrEqual** Only choose data for which the value on the left is greater than or equal to the value on the right.
- **LessThan** Only choose data for which the value on the left is less than the value on the right.
- **LessThanOrEqual** Only choose data for which the value on the left is less than or equal to the value on the right.
- **TopN** Only choose items from the value on the left which are the top number specified in the value on the right.
- **BottomN** Only choose items from the value on the left which are the bottom number specified in the value on the right.
- **TopPercent** Only choose items from the value on the left which are the top percent specified in the value on the right.
- **BottomPercent** Only choose items from the value on the left which are the bottom percent specified in the value on the right.
- **In** Only choose items from the value on the left which are in the array of values specified on the right. Selecting this operator enables the Values list at the bottom.
- **Between** Only choose items from the value on the left which fall between the pair of values you specify on the right. Selecting this operator enables two Value boxes instead of one.

Value: Enter a value to compare with the expression on the left based on the selected operator. For multiple values used with the **Between** operator, the lower two value boxes are enabled.

Values: When you choose the **In** operator, you can enter as many values as you need in this list.

Sorting

Click the plus sign button to enter new sort expressions, and remove them using the X button.

In the **Expression** box, enter an expression by which to sort the data in the group, and under **Direction**, select **Ascending** or **Descending** for the selected sort expression.

Data Output

Element name: Enter a name to be used in the XML output for this sparkline report control.

Output: Choose **Auto**, **Yes**, **No, Contents Only** to decide whether to include this Sparkline in the XML output. Choosing **Auto** exports the contents of the Sparkline report control.

Subreport

The Subreport control is a placeholder for data from a separate report. In ActiveReports, for better performance, we recommend using data regions instead of Subreport controls wherever possible. The reason is that the report server must process every instance of each subreport, which can become burdensome in very large reports with a large number of subreports processed many times per report. Using data regions to display separate groups of data can be much more efficient in such reports. For more information, see Report Controls.

> **Note**:

- A Page report can use an RDL report as the target subreport.
- You cannot use a Section report as the target of a Subreport in a Page/RDL report, and vice versa.

Subreports make sense when you need to nest groups of data from different data sources within a single data region, or when you can reuse a subreport in a number of reports. Here are some things to keep in mind while designing subreports.

- If you make changes to the subreport without changing the main report, click **Refresh** to re-run the subreport in the Preview tab.
- If you design the parent report in a stand-alone Report Designer application, save the parent report to the same directory as the subreport to render it in the Preview tab.
- If you set borders on the Subreport control and the body of the report hosted in it, ActiveReports does not merge the two borders.
- If the report hosted in the Subreport control cannot be found or contains no rows, only the border of the Subreport control is rendered.
- If the hosted report is found and does contain rows, the border of the hosted report body is rendered.

Parameters

You can use parameters supplied by the parent report to filter data displayed in a subreport. You can also pass parameters to a repeating subreport nested in a data region to filter each instance.

Subreport Dialog

Properties for the Subreport are available in the Subreport dialog. To open it, with the Subreport control selected on the report, under the Properties Window, click the **Property dialog** link.

The Subreport dialog lets you set properties on the report control with the following pages.

Note: You can select **<Expression...>** within any of these properties to open the Expression Editor.

General

Name: Enter a name for the subreport that is unique within the report. This name can be called in code. You can only use underscore (_) as a special character in the Name field. Other special characters such as period (.), space ( ), forward slash (/), back slash (\), exclamation (!), and hyphen (-) are not supported.

Tooltip: A textual label for the report item used to include TITLE or ALT attributes in HTML reports.

Subreport:
Select the **<From File...>** option to open the **Open** dialog, and then select a report to display within the Subreport control.

Use this report's theme when rendering subreport: Select this check box to have the subreport automatically use the same theme as the hosting report.

Visibility

Initial visibility

- **Visible**: The subreport is visible when the report runs.
- **Hidden**: The subreport is hidden when the report runs.
- **Expression**: Use an expression with a Boolean result to decide whether the subreport is visible. True for hidden, false for visible.

Visibility can be toggled by another report control: Select this check box to display a toggle image next to another report control. This enables the drop-down box where you can specify the TextBox control which, if clicked, toggles the visibility of the subreport.

Navigation

Document map label: Enter an expression to use as a label to represent this item in the table of contents (document map).

Bookmark ID: Enter an expression to use as a locator for this subreport. You will then be able to provide a bookmark link to this item from another report control using a **Jump to bookmark** action.

Parameters

The Parameters page of the Subreport dialog allows you to enter new parameters and remove or change the order of parameters using the X and arrow buttons. For each parameter in this list, there is a **Parameter Name** and a **Parameter Value**.

Each **Parameter Name** must exactly match the name of a parameter in the target report.

For the **Parameter Value**, enter an expression to use to send information from the summary or main report to the subreport target.

> **Note**: The following metods are not supported in the **Parameter Value** expression for a parameter that passes a value from the main report to subreport. However, this restriction does not apply to parameter default values.
>
> - RowNumber
> - RunningValue
> - Lookup/LookupSet
> - Previous
> - CountRows
> - CumulativeTotal

Data Output

Element name: Enter a name to be used in the XML output for this subreport.

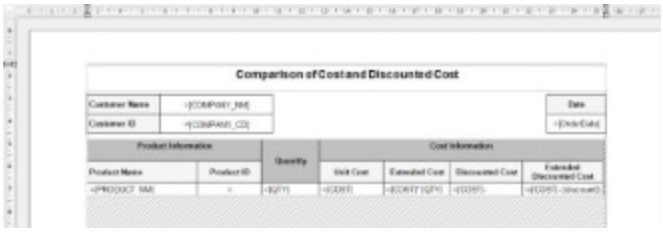
Output: Choose **Auto**, **Yes**, or **No** to decide whether to include this subreport in the XML output. Choosing **Auto** exports the contents of the subreport.

Table

The Table data region consists of columns and rows that organize data. A Table has three columns and three rows by default, a total of nine cells, each of which is filled with a text box. At design time, you can add or remove columns, rows and groupings to suit your needs. In Page and RDL reports, you can embed other data regions in table cells.

You can also choose to set the height of multiple rows or width of multiple columns using **Distribute Rows Evenly** and **Distribute Columns Evenly** options from the context menu of the Table data region. Multiple rows or columns can be selected using **Ctrl** key and mouse click combination or by simply dragging the mouse over rows and columns.

You can merge cells in both of horizontal and vertical direction in the Header, Group Header, Footer, Group Footer, and Detail sections when creating a report in ActiveReports Designer and ActiveReports Web Designer. You can merge cells inside one section by using **Merge Cells** in the context menu of selected cells. Cells of different sections cannot be merged with each other.



Adding Data

Once you place the Table data region on a report, you can add data to its cells. As with any data region, you can drag fields from your Fields list onto cells in the table. Although the default report control within each cell of the table is a text box, you can replace it with any other report control, and on RDL reports, you can even add a data region. When you drag a field into a cell in the detail row, ActiveReports automatically provides a label in the table header. As with all report controls, you can use expressions to further manipulate the data within the cells of the table. For more information, see Expressions.

Grouping

One of the types of rows you can add to the table is the group header or group footer. For example, if you have a report that displays Customer details – CustomerID, Address, and Country Name fields, you can add a group header, and in the **Group On** expression, choose the Country Name field to group the details based on country name.

You can also group the data in your detail section, and you can add multiple rows to any group in the table. You can use aggregate functions in group footer rows to provide subtotals. For more information, see Group Data.

Appearance

There are many ways in which you can control the appearance of the table data region. You can merge cells, control visibility, text color, and background color. You can use graphical elements within the cells and borders on the cells themselves. For more information on freezing row and column headers in RDL reports, see Freeze Rows and Columns.

Table Dialog

Properties for the Table are available in the Table dialog. To open it, with the Table data region selected on the report, under the Properties Window, click the **Property dialog** link.

The Table dialog lets you set properties on the report control with the following pages.

> **Note**: You can select **<Expression...>** within these properties to create an expression to determine the value.

General

Name: Enter a name for the table that is unique within the report. This name can be called in code. You can only use underscore (_) as a special character in the Name field. Other special characters such as period (.), space ( ), forward slash (/), back slash (\), exclamation (!), and hyphen (-) are not supported.

Tooltip: A textual label for the report item used to include TITLE or ALT attributes in HTML reports.

Dataset name: Select a dataset to associate with the table. The combo box is populated with all of the datasets in the report's dataset collection.

Header and Footer: Select any of the following options.

- Repeat header row on each page

- Repeat footer row on each page
- Prevent orphaned footer on next page

Visibility

Initial visibility

- **Visible**: The table is visible when the report runs.
- **Hidden**: The table is hidden when the report runs.
- **Expression**: Use an expression with a Boolean result to decide whether the table is visible. True for hidden, false for visible.

Visibility can be toggled by another report control: Select this check box to display a toggle image next to another report control. This enables the drop-down box where you can specify the report control which, if clicked, toggles the visibility of the table.

Navigation

Document map label: Enter an expression to use as a label to represent this item in the table of contents (document map).

Bookmark ID: Enter an expression to use as a locator for this table. You will then be able to provide a bookmark link to this item from another report control using a **Jump to bookmark** action.

Sorting

Click the plus sign button to enter new sort expressions, and remove them using the X button.

In the **Expression** box, enter an expression by which to sort the data in the group, and under **Direction**, select **Ascending** or **Descending** for the selected sort expression.

Groups

The Groups page of the Table dialog allows you to remove or change the order of items in the Group list using the X and arrow buttons. Click the **Add** button to add a new group to the table and set up information for each group on the following tabs.

General

Name: Enter a name for the group that is unique within the report. This property cannot be set until after a Group on expression is supplied.

Group on: Enter an expression to use for grouping the data.

Document map label: Enter an expression to use as a label to represent this item in the table of contents (document map).

Parent group: For use in recursive hierarchies. Enter an expression to use as the parent group.

Filters

You need to provide three values to add a new filter to the collection: Expression, Operator, and Value.

Expression: Enter the expression to use for evaluating whether data should be included in the group.

Operator: Select from the following operators to decide how to compare the expression to the left with the value to the right.

- **Equal** Only choose data for which the value on the left is equal to the value on the right.
- **Like** Only choose data for which the value on the left is similar to the value on the right. For more information on using the **Like** operator, see the MSDN Web site.
- **NotEqual** Only choose data for which the value on the left is not equal to the value on the right.
- **GreaterThan** Only choose data for which the value on the left is greater than the value on the right.
- **GreaterThanOrEqual** Only choose data for which the value on the left is greater than or equal to the value on the right.
- **LessThan** Only choose data for which the value on the left is less than the value on the right.
- **LessThanOrEqual** Only choose data for which the value on the left is less than or equal to the value on the right.
- **TopN** Only choose items from the value on the left which are the top number specified in the value on the right.
- **BottomN** Only choose items from the value on the left which are the bottom number specified in the value on the right.
- **TopPercent** Only choose items from the value on the left which are the top percent specified in the value on the right.
- **BottomPercent** Only choose items from the value on the left which are the bottom percent specified in the value on the right.
- **In** Only choose items from the value on the left which are in the array of values specified on the right. Selecting this operator enables the Values list at the bottom.
- **Between** Only choose items from the value on the left which fall between the pair of values you specify on the right. Selecting this operator enables two Value boxes instead of one.

Value: Enter a value to compare with the expression on the left based on the selected operator. For multiple values used with the **Between** operator, the lower two value boxes are enabled.

Values: When you choose the **In** operator, you can enter as many values as you need in this list.

Sorting

Click the plus sign button to enter new sort expressions, and remove them using the X button.

In the **Expression** box, enter an expression by which to sort the data in the group, and under **Direction**, select **Ascending** or **Descending** for the selected sort expression.

Visibility

Initial visibility

- **Visible**: The group is visible when the report runs.
- **Hidden**: The group is hidden when the report runs.
- **Expression**: Use an expression with a Boolean result to decide whether the group is visible. True for hidden, False for visible.

Visibility can be toggled by another report control: Select this check box to display a toggle image next to another report control. The user can click the toggle item to show or hide this band group. This enables the drop-down list where you can select the TextBox control that users can click to show or hide this group.

Data Output

Element name: Enter a name to be used in the XML output for this group.

Collection: Enter a name to be used in the XML output for the collection of all instances of this group.

Output: Choose **Yes** or **No** to decide whether to include this group in the XML output.

Layout

Page break at start: Inserts a page break before the group.

Page break at end: Inserts a page break after the group.

Include group header: Adds a group header band (selected by default).

Include group footer: Adds a group footer band (selected by default).

Repeat group header: Repeats the group header band on each page.

Repeat group footer: Repeats the group footer band on each page.

Prevent orphaned footer: Prints the last detailed row with the footer in order to prevent orphaned footer on the next page.

Detail Grouping

Detail grouping is useful when you do not want to repeat values within the details. When a detail grouping is set, the value repeats for each distinct result of the grouping expression instead of for each row of data. For example, if you use the Customers table of the NorthWind database to create a list of countries without setting the details grouping, each country is listed as many times as there are customers in that country. If you set the details grouping to `=Fields!Country.Value` each country is listed only once.

> **Note**: If the detail grouping expression you use results in a value that is distinct for every row of data, a customer number for example, you will see no difference in the results.

Go to the Detail Grouping page has the following tabs.

General

Name: Enter a name for the group that is unique within the report. This property cannot be set until after a Group on expression is supplied.

Group on: Enter an expression to use for grouping the data.

Document map label: Enter an expression to use as a label to represent this item in the table of contents (document map).

Parent group: For use in recursive hierarchies. Enter an expression to use as the parent group.

Filters

You need to provide three values to add a new filter to the collection: Expression, Operator, and Value.

Expression: Enter the expression to use for evaluating whether data should be included in the group.

Operator: Select from the following operators to decide how to compare the expression to the left with the value to the right.

- **Equal** Only choose data for which the value on the left is equal to the value on the right.
- **Like** Only choose data for which the value on the left is similar to the value on the right.
  For more information on using the **Like** operator, see the MSDN Web site.
- **NotEqual** Only choose data for which the value on the left is not equal to the value on the right.
- **GreaterThan** Only choose data for which the value on the left is greater than the value on the right.
- **GreaterThanOrEqual** Only choose data for which the value on the left is greater than or equal to the value on the right.
- **LessThan** Only choose data for which the value on the left is less than the value on the right.
- **LessThanOrEqual** Only choose data for which the value on the left is less than or equal to the value on the right.
- **TopN** Only choose items from the value on the left which are the top number specified in the value on the right.

- **BottomN** Only choose items from the value on the left which are the bottom number specified in the value on the right.
- **TopPercent** Only choose items from the value on the left which are the top percent specified in the value on the right.
- **BottomPercent** Only choose items from the value on the left which are the bottom percent specified in the value on the right.
- **In** Only choose items from the value on the left which are in the array of values specified on the right. Selecting this operator enables the Values list at the bottom.
- **Between** Only choose items from the value on the left which fall between the pair of values you specify on the right. Selecting this operator enables two Value boxes instead of one.

Value: Enter a value to compare with the expression on the left based on the selected operator. For multiple values used with the **Between** operator, the lower two value boxes are enabled.

Values: When you choose the **In** operator, you can enter as many values as you need in this list.

Visibility

Initial visibility

- **Visible**: The group is visible when the report runs.
- **Hidden**: The group is hidden when the report runs.
- **Expression**: Use an expression with a Boolean result to decide whether the group is visible. True for hidden, False for visible.

Visibility can be toggled by another report control: Select this check box to display a toggle image next to another report control. The user can click the toggle item to show or hide this band group. This enables the drop-down list where you can select the report control that users can click to show or hide this group.

Data Output

Element name: Enter a name to be used in the XML output for this group.

Collection: Enter a name to be used in the XML output for the collection of all instances of this group.

Output: Choose **Yes** or **No** to decide whether to include this group in the XML output.

Layout

Page break at start: Inserts a page break before the group.

Page break at end: Inserts a page break after the group.

Has own page numbering: Used in conjunction with the "Page Number in Section" and "Total Pages in Section" properties, tells the report that the group constitutes a new page numbering section.

Filters

You need to provide three values to add a new filter to the collection: Expression, Operator, and Value.

Expression: Enter the expression to use for evaluating whether data should be included in the table.

Operator: Select from the following operators to decide how to compare the expression to the left with the value to the right:

- **Equal** Only choose data for which the value on the left is equal to the value on the right.
- **Like** Only choose data for which the value on the left is similar to the value on the right.

For more information on using the **Like** operator, see the MSDN Web site.

- **NotEqual** Only choose data for which the value on the left is not equal to the value on the right.
- **GreaterThan** Only choose data for which the value on the left is greater than the value on the right.
- **GreaterThanOrEqual** Only choose data for which the value on the left is greater than or equal to the value on the right.
- **LessThan** Only choose data for which the value on the left is less than the value on the right.
- **LessThanOrEqual** Only choose data for which the value on the left is less than or equal to the value on the right.
- **TopN** Only choose items from the value on the left which are the top number specified in the value on the right.
- **BottomN** Only choose items from the value on the left which are the bottom number specified in the value on the right.
- **TopPercent** Only choose items from the value on the left which are the top percent specified in the value on the right.
- **BottomPercent** Only choose items from the value on the left which are the bottom percent specified in the value on the right.
- **In** Only choose items from the value on the left which are in the array of values specified on the right. Selecting this operator enables the Values list at the bottom.
- **Between** Only choose items from the value on the left which fall between the pair of values you specify on the right. Selecting this operator enables two Value boxes instead of one.

Value: Enter a value to compare with the expression on the left based on the selected operator. For multiple values used with the **Between** operator, the lower two value boxes are enabled.

Values: When you choose the **In** operator, you can enter as many values as you need in this list.

Data Output

Element name: Enter a name to be used in the XML output for this table.

Output: Choose **Auto**, **Yes**, or **No** to decide whether to include this table in the XML output. Choosing **Auto** exports the contents of the table.

Detail element name: Enter a name to be used in the XML output for the data element for instances of the table. This name is ignored if you have specified a details grouping.

Detail collection name: Enter a name to be used in the XML output for the data element for the collection of all instances of the detail grouping.

Data element output: Choose **Yes** or **No** to decide whether to include the details in the XML output.

Table Layout Actions

The Table data region provides context menu options to perform basic layout actions. You can access layout options for Table rows from the context menu by right-clicking on a selected row.

- **Insert Row Above:** Add a row above the selected row. The inserted row type is similar to the type of selected row, that is, if the selected row type is a header row, a new header row is added.
- **Insert Row Below:** Add a row below the selected row. The inserted row type is similar to the type of selected row, that is, if the selected row type is a header row, a new header row is added.
- **Delete Rows:** Delete the selected rows.
- **Distribute Rows Evenly:** Set the same height for multiple selected rows.
- **Table Header:** Show or hide table header rows.
- **Table Details:** Show or hide table detail rows.
- **Table Footer:** Show or hide table footer rows.
- **Insert Group:** Insert groups in a table.

- **Edit Group:** Edit a group in a table.
- **Delete Groups:** Delete groups in a table.

You can access layout options for Table columns from the context menu by right-clicking on a selected column.

- **Insert Column to the Left:** Add a column to the left of the selected column.
- **Insert Column to the Right:** Add a column to the right of the selected column.
- **Distribute Columns Evenly:** Set the same width for multiple selected columns.
- **Add Columns**: Add one or more column(s) to the right of the selected column.
- **Delete Columns**: Delete the selected columns.

Table of Contents

The TableOfContents report control is used to display the document map, an organized hierarchy of the report heading levels and labels along with their page numbers, in the body of a report. The TableOfContents control allows you to quickly understand and navigate the data inside a report in all viewers that are supported in ActiveReports. Unlike the Document Map that is only available in the Viewers and cannot be rendered or printed, you can use the TableOfContents control to embed the TableOfContents structure in the report body for printing and rendering purposes. You can add a TableOfContents report control to a report by dragging it from the toolbox and dropping it onto the report design surface.

In the Properties Window, there are a number of properties that you can use to control the appearance and behavior of the TableOfContents report control. For example, you can use the **OverflowName** property to specify the OverflowPlaceHolder control name to link it with the TableOfContents control. The **FixedHeight** property allows you to set the maximum height of the TableOfContents control on each page, similar to the FixedSize property that is available with other report controls. The **StyleName** property allows you to apply the selected styles from a style sheet. These styles can be applied to the TableOfContents report control using the **StyleName** property or to Table Of Contents levels using the **LevelDesigner Collection Editor** dialog. For more information on how to set styles, see Add TableofContents.

The **Levels** property contains the collection of TableOfContents levels and allows you to access the **LevelDesigner Collection Editor** dialog, where you can set up the report TableOfContents levels and their properties. The **MaxLevel** property restricts the maximum number of levels in the document map.

Any customization made to the Document Map like setting **Numbering Style** for document map levels using the Report dialog or using the **DocumentMap** property gets directly applied to the TableOfContents control. For more information, see Add Items to the Document Map.

Table Of Contents properties Dialog

Properties for the TableOfContents are available in the Table Of Contents dialog. To open it, with the TableOfContents control selected on the report, under the Properties Window, click the **Property dialog** link.

The Table Of Contents Properties dialog lets you set properties on the report control with the following pages.

> **Note**: You can click <**Expression...**> in many of these properties to open the Expression Editor where you can create an expression to determine the value.

General

Name: Enter a name for the table of contents that is unique within the report. This name can be called in code. You can only use underscore (_) as a special character in the Name field. Other special characters such as period (.), space ( ),

forward slash (/), back slash (\), exclamation (!), and hyphen (-) are not supported.

Tooltip: A textual label for the report item used to include TITLE or ALT attributes in HTML reports.

Visibility

By default, the TableOfContents is visible when the report runs, but you can hide it, hide it only when certain conditions are met, or toggle its visibility with another report control.

Initial visibility

Visible: The TableOfContents is visible when the report runs.

Hidden: The TableOfContents is hidden when the report runs.

Expression: Use an expression with a Boolean result to decide whether the TableOfContents is visible. True for hidden, false for visible.

Visibility can be toggled by another report control: Select this check box to display a toggle TableOfContents next to another report control. This enables the drop-down box where you can specify the TextBox control which, if clicked, toggles the visibility of the TableOfContents.

Appearance

Border

Style: Select a style for the border.

Width: Enter a value in points to set the width of the border.

Color: Select a color to use for the border, or select the **<Expression...>** option to open the Expression Editor and create an expression that evaluates to a .NET color.

Background

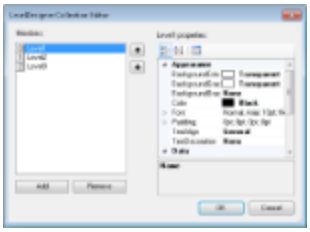
Color: Select a color to use for the background.

Data Output

Element name: Enter a name to be used in the XML output for the TableOfContents report control.

Output: Choose **Auto**, **Yes**, **No** to decide whether to include this TableOfContents in the XML output. Choosing **Auto** exports the contents of the TableOfContents report control.

LevelDesigner Collection Editor



The LevelDesigner Collection Editor is used to set up the report TableOfContents levels and their properties. To access the **LevelDesigner Collection Editor** dialog, go to the Properties Window and in the **Levels** property, click (Collection).

You can set the TableOfContents level properties from the following locations:

- The Properties grid of the LevelDesigner Collection Editor.
- The Level properties dialog that gets displayed if you click the **Property Pages** button above the LevelDesigner Collection Editor Properties grid.

The LevelDesigner Collection Editor lets you set the properties of a TableOfContents level as follows.

Appearance

BackgoundColor: Select a color to use for the background of the TableOfContents level.

Color: Select the color of the text.

Font: Select the font to render the TableOfContents level text.

Style: Choose Normal, Italic or select the <Expression...> option to open the Expression Editor and create an expression.

Family: Choose the font family name.

Size: Choose the size in points for the font.

Weight: Choose from Lighter, Thin, ExtraLight, Light, Normal, Medium, SemiBold, Bold, ExtraBold, Heavy, and Bolder, or select the <Expression...> option to open the Expression Editor and create an expression.

Padding: Specify left, right, top and bottom values for the padding to apply to a TableOfContents level.

StyleName: Select a style to apply to the TableOfContents level.

TextAlign: Specify the horizontal alignment of the text.

TextDecoration: Choose from None, Underline, Overline, and LineThrough, or select the <Expression...> option to open the Expression Editor and create an expression.

Data

DataElementName: Enter a name to be used in the XML output for this TableOfContents level.

General

DisplayFillCharacters: Specifies whether to display a leading character. The Default value is **True**.

DisplayPageNumber: Specifies whether to display a page number. The Default value is **True**.

FillCharacter: Use the expression to specify a fill character for a leading character.

Layout

TextIndent: Specify the text indent.

Misc

Name: Specify a name for the TableOfContents level.

TextBox

The Textbox is the most commonly used report control that displays data. By default, the TextBox appears in each cell of a Table or Tablix data region. Also, the TextBox is what is created when you drag a field from the Data Explorer onto the

report.

In the **Value** property of the TextBox, you can enter static text or an expression. An expression can display fields from a database, calculate a value, or visually display data.

> **Tip**: You can enter text directly into the TextBox on the design surface of the report by double-clicking inside it.

In the Properties Window are a number of properties that you can use to control the appearance and behavior of the TextBox. For example, you can set the **Action** property to have the viewer jump to a bookmark within the report, another report, or a URL when a user clicks the TextBox at run time. The **DataElement** properties allow you to control how and whether the TextBox displays in XML exports.

By default, in RDL Reports, the TextBox can grow vertically to accommodate the data it displays, and it cannot shrink smaller than it appears at design time. To change this behavior, set the **CanShrink** and **CanGrow** properties in the Properties grid. (These properties are not available in Page Reports.)

Data Fields

When you drag a field from a dataset in the Data Explorer and drop it onto the report surface, a TextBox report control with an expression is automatically created. The type of expression that is created depends upon the context where you drop the field. The following table describes the various contexts and expressions created if you drag a field named **SalesAmount** onto the report.

Expressions created for fields in different contexts

> **Note**: The expression created is different for a field with a string or unknown data type. In these cases, the **First** aggregate is used in place of the **Sum** aggregate in the expressions below. At run time, the first value found within the scope is displayed instead of a summary.

Context	Expression	Run-Time Behavior
Directly on the report surface	=Sum(Fields!SalesAmount.Value)	Displays a summary of the sales amount for the entire dataset.
List data region	=Fields!SalesAmount.Value	Displays a value for each row of data, in a list running down the page.
BandedList data region, header or footer band	=Sum(Fields!SalesAmount.Value)	Displays a summary of the sales amount for the dataset associated with the BandedList.
BandedList data region, detail band	=Fields!SalesAmount.Value	Displays a value for each row of data, in a list running down the page.
BandedList data region, group header or footer band	=Sum(Fields!SalesAmount.Value)	Displays a summary of the sales amount for the grouping.
Table data region, header or footer row	=Sum(Fields!SalesAmount.Value)	Displays a summary of the sales amount for the dataset associated with the Table.
Table data region, detail row	=Fields!SalesAmount.Value	Displays a value for each row of data, in a list running down the page.
Table data region, group header or footer row	=Sum(Fields!SalesAmount.Value)	Displays a summary of the sales amount for the grouping.

Tablix data region, corner cell	none	Displays a blank cell. You can add a label or even use this area to embed other report control.
Tablix data region, column group cell	=Fields!SalesAmount.Value	Displays the value at the top of a new column for each row of data running to the right.
Tablix data region, row group cell	=Fields!SalesAmount.Value	Displays the value to the left of a new row for each row of data running down the page.
Tablix data region, body cell	=Sum(Fields!SalesAmount.Value)	Displays a summary of the sales amount for the intersection of the column and row.

Textbox Dialog

Properties for the Textbox are available in the Textbox dialog. To open it, with the Textbox control selected on the report, under the Properties Window, click the **Property dialog** link.

The Textbox dialog lets you set properties on the report control with the following pages.

> **Note**: You can select **<Expression...>** within many of these properties to open the Expression Editor. You can also access the **Expression Editor** from the context menu of the TextBox control.

General

Name: Enter a name for the textbox that is unique within the report. This name is displayed in the Document Outline and in XML exports. You can only use underscore (_) as a special character in the Name field. Other special characters such as period (.), space ( ), forward slash (/), back slash (\), exclamation (!), and hyphen (-) are not supported.

Tooltip: A textual label for the report item used to include TITLE or ALT attributes in HTML reports.

> **Note**: When the group or dataset breaks to a new page, the first instance of the repeated value is printed.

Visibility

Initial visibility allows you to select from the following options:

- **Visible**: The textbox is visible when the report runs.
- **Hidden**: The textbox is hidden when the report runs.
- **Expression**: Use an expression with a Boolean result to decide whether the textbox is visible. For example, on a "Free Shipping" textbox, you could use the expression to see whether the ShippingCountry is international. A value of True hides the textbox, False shows it.

Visibility can be toggled by another report control: If you select this check box, it enables the drop-down box where you can specify the TextBox control that users can click to toggle the visibility of the textbox.

Initial appearance of the toggle image: allows you to select from the following options:

- **Expanded**: The toggle image shows as a minus sign, and all instances of this textbox are visible.
- **Collapsed**: The toggle image shows as a plus sign, and all instances of this textbox are hidden.
- **Expression**: Use an expression with a Boolean result to decide whether the toggle image is expanded. A value of True expands the toggle image, False collapses it.

Navigation

Action

Select one of the following actions to perform when a user clicks on the textbox.

None: The default behavior is to do nothing when a user clicks the textbox at run time.

Jump to report: For drill-through reporting, select this option and provide the name of a local report, the relative path of a report in another folder, or the full path of a report on another server. You can also use expressions to create drill-through links.

Parameters: Supply parameters to the targeted report by entering the **Name** of each parameter, the **Value** to send to the targeted report, or whether to **Omit** the parameter. Note that parameter names you supply must exactly match parameters in the target report. You can remove or change the order of parameters using the X and arrow buttons.

Jump to bookmark: Select this option and provide a valid Bookmark ID to allow the user to jump to the report control with that Bookmark ID.

Jump to URL: Select this option and provide a valid URL to create a hyperlink to a Web page.

Document map label: Enter an expression to use as a label to represent this item in the table of contents (document map).

Bookmark ID: Enter an expression to use as a locator for this textbox. You will then be able to provide a bookmark link to this item from another report control using a **Jump to bookmark** action.

Appearance

Border

Style: Select a style for the border.

Width: Enter a value in points to set the width of the border.

Color: Select a color to use for the border, or select the **<Expression...>** option to open the Expression Editor and create an expression that evaluates to a .NET color.

Background

Color: Select a color to use for the background of the textbox.

Image: Enter an image to use for the background of the textbox.

> **Note**: The Background Color and Background Image properties allow you to choose the **<Data Visualizer...>** option as well to launch the dialog that let you build a data visualization expression.

Font

Family: Select a font family name or a theme font.

Size: Choose the size in points for the font or use a theme.

Style: Choose **Normal** or **Italic** or select a theme.

Weight: Choose from **Lighter**, **Thin**, **ExtraLight**, **Light**, **Normal**, **Medium**, **SemiBold**, **Bold**, **ExtraBold**, **Heavy**, or **Bolder**.

Color: Choose a color to use for the text.

Decoration: Choose from **None**, **Underline**, **Overline**, or **LineThrough**.

Format

Format code: Select one of the common numeric formats provided or use a custom .NET formatting code to format dates or numbers. For more information, see MSDN's Formatting Types topic.

Line Spacing: This property sets the space between lines of text.

Character Spacing: This property sets the space between characters.

Minimal rate of text horizontal shrinking (in %): Specify the percentage to which the text should be shrunk horizontally.

Textbox height

Can increase to accommodate contents: Select this check box to set CanGrow to True.

Can decrease to accommodate contents: Select this check box to set CanShrink to True.

Can shrink text to fit fixed size control: Select this check box to set ShrinkToFit to True.

Text direction and writing mode

Direction: Choose **LTR** for left to right, or **RTL** for right to left.

Mode: Choose **lr-tb** for left right top bottom (normal horizontal text) or **tb-rl** for top bottom right left (vertical text on its side).

Angle: Enter the number of degrees to rotate the text in a counter-clockwise direction. Enter a negative number to rotate the text in a clockwise direction.

Alignment

Vertical alignment: Choose **Top**, **Middle**, **Bottom**, or the **<Expression...>** option.

Horizontal alignment: Choose **General**, **Left**, **Center**, **Right**, **Justify**, or the **<Expression...>** option.

Justify method: Set Horizontal alignment to **Justify** to enable this property. Choose **Auto**, **Distribute**, **DistributeAllLines**, or the **<Expression...>** option.

Wrap mode: Choose **NoWrap**, **WordWrap**, or **CharWrap**.

Amount of space to leave around report control

- **Top padding**: Set the top padding in points.
- **Left padding**: Set the left padding in points.
- **Right padding**: Set the right padding in points.
- **Bottom padding**: Set the bottom padding in points.

Interactive Sort

Select the checkbox next to **Add an interactive sort action to this textbox** to enable the following controls which allow end users to sort the report data in the viewer.

Sort expression: Enter an expression to use for determining the data to sort.

Data region or group to sort: Select the grouping level or data region within the report to sort. The default value is **Current scope**, but you may also elect to choose an alternate data region or grouping.

Evaluate sort expression in this scope: Select the grouping level within the report on which to evaluate an aggregate sorting expression. The default value is **Current scope**, but you may also elect to choose an alternate data region or grouping.

Data Output

Element Name: Enter a name to be used in the XML output for this textbox.

Output: Choose **Auto**, **Yes**, or **No** to decide whether to include this textbox in the XML output. **Auto** exports the contents of the textbox only when the value is not a constant.

Render as: Choose **Auto**, **Element**, or **Attribute** to decide whether to render textboxes as Attributes or Elements in the exported XML file. **Auto** uses the report's setting for this property.

> **Attribute example**: <table1 textbox3="Report created on: 7/26/2005 1:13:00 PM">
>
> **Element example**: <table1> <textbox3>Report created on: 7/26/2005 1:13:28 PM</textbox3>

Tablix

A Tablix data region displays data in cells that are arranged in rows and columns. It provides enhanced layout capabilities ranging from creation of simple tables to advanced matrices. Tablix is essentially a combination of two data regions, the table and the matrix. Therefore, it provides all the features of a table and a matrix along with added capabilities including support for multiple adjacent groups on rows or columns and improved layout flexibility with stepped group layouts.

You can also choose to set the height of multiple rows or width of multiple columns using **Distribute Rows Evenly** and **Distribute Columns Evenly** options from the context menu of the Tablix data region. You can select multiple rows or columns using the **Ctrl** key and mouse click together or by simply dragging the mouse over rows and columns. For more information on freezing row and column headers in RDL reports, see Freeze Rows and Columns.

This topic describes how the elements of the Tablix data region work together and explains its basic operations.

- **Areas of Tablix Data Region**
- **Row Column Handles**
- **Tablix Layout Actions**

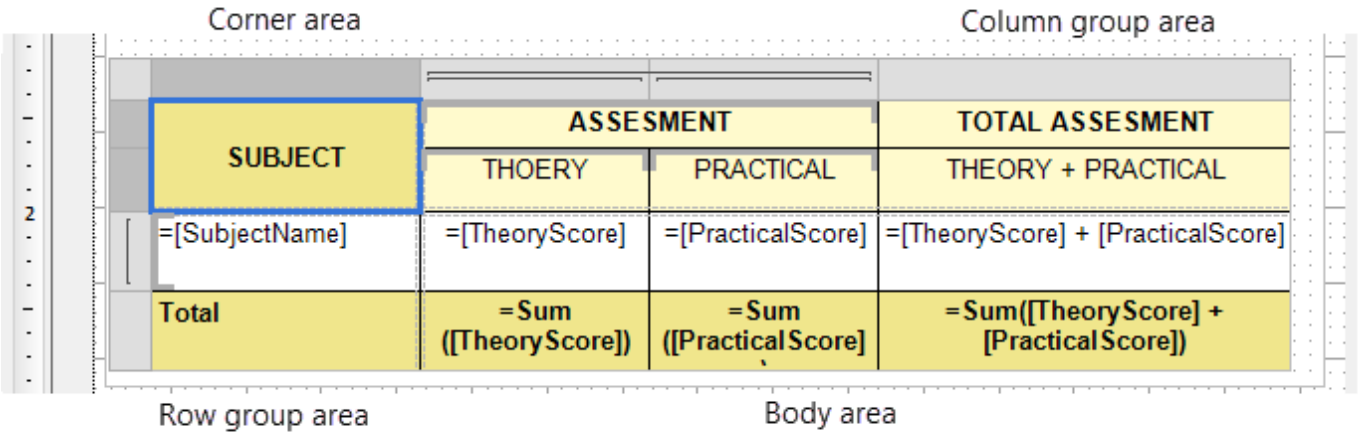
Areas of the Tablix Data Region

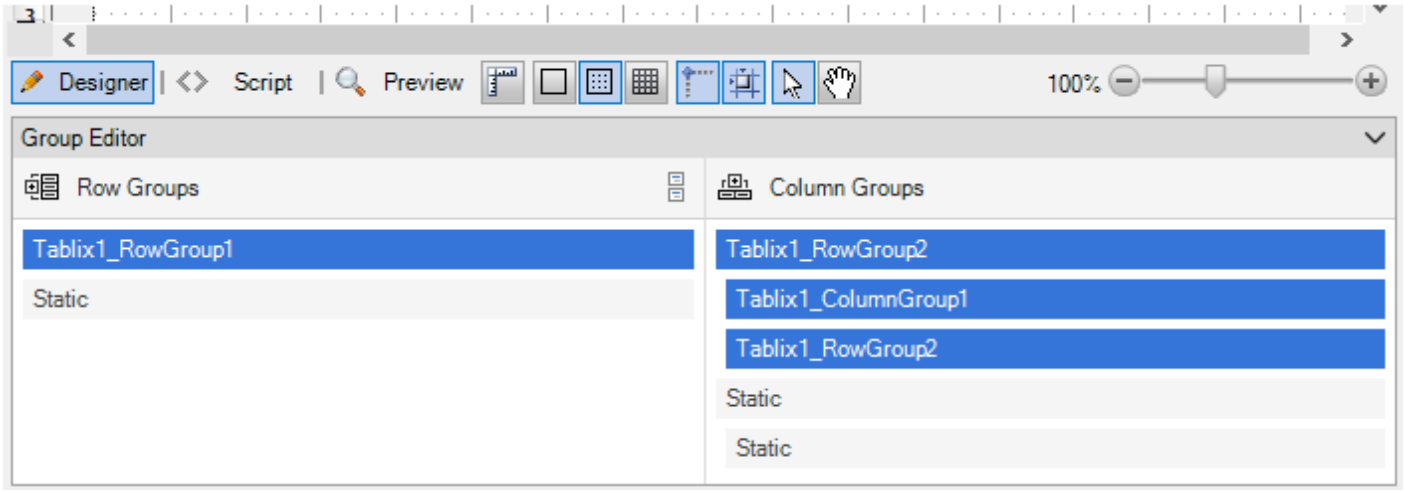
The Tablix data region is composed of four areas denoted by dotted lines on the design surface: the corner, the row group area, the column group area, and the body. By default, each tablix cell contains a TextBox control and the function for each cell is determined by its location. You can change the layout of the Tablix data region using the **GrapeCity.ActiveReports.PageReportModel.Tablix.LayoutDirection** property.

ActiveReports includes a **Group Editor** window that is specifically designed to manage the tablix structure. In the Visual Studio integrated designer as well as in applications using the Designer control, the Group Editor window is located below the report design surface. Developers can use it in custom designer applications as well.

> **Note:** In case the **Group Editor** window does not appear automatically in your application, select **View > Other Windows > Group Editor 13** in your Visual Studio project.

The image below demonstrates the areas of a Tablix data region, where column groups are set to **TheoryScore** and **PracticalScore**, and the row group is set to **SubjectName**.





The Group Editor window contains the following groups:

- **Row Groups:** The Row Groups section in the Group Editor displays all the groups that are applied in the row group area of the Tablix data region.

  Subject
  The left center cell of the Tablix data region **=[SubjectName]** represents the **Subject** group in the Group Editor window. This group displays the subject names for the current semester.

- **Column Groups :** The Column Groups section in the Group Editor window displays all the groups that are applied in the column group area of the Tablix data region.

  Theory
  The center left cell of the Tablix data region **=[TheoryScore]** represents the **Theory** group in the Group Editor window. This group displays theory scores for the student in each subject.

  Practical
  The center right cell of the Tablix data region **=[PracticalScore]** represents the **Practical** group in the Group Editor window. This group displays practical scores for the student in each subject.

- **Static Cells:** Static cells in the Row Groups and Column Groups are not represented in the Group Editor window

because these cells are not associated with any grouped data. Static row and column cells are used to display labels and totals in a Tablix data region.

The static column cell displays the label **ASSESSMENT** and **TOTAL ASSESSMENT** in the Tablix data region. The static row cell displays the label **Total** in the Tablix data region.

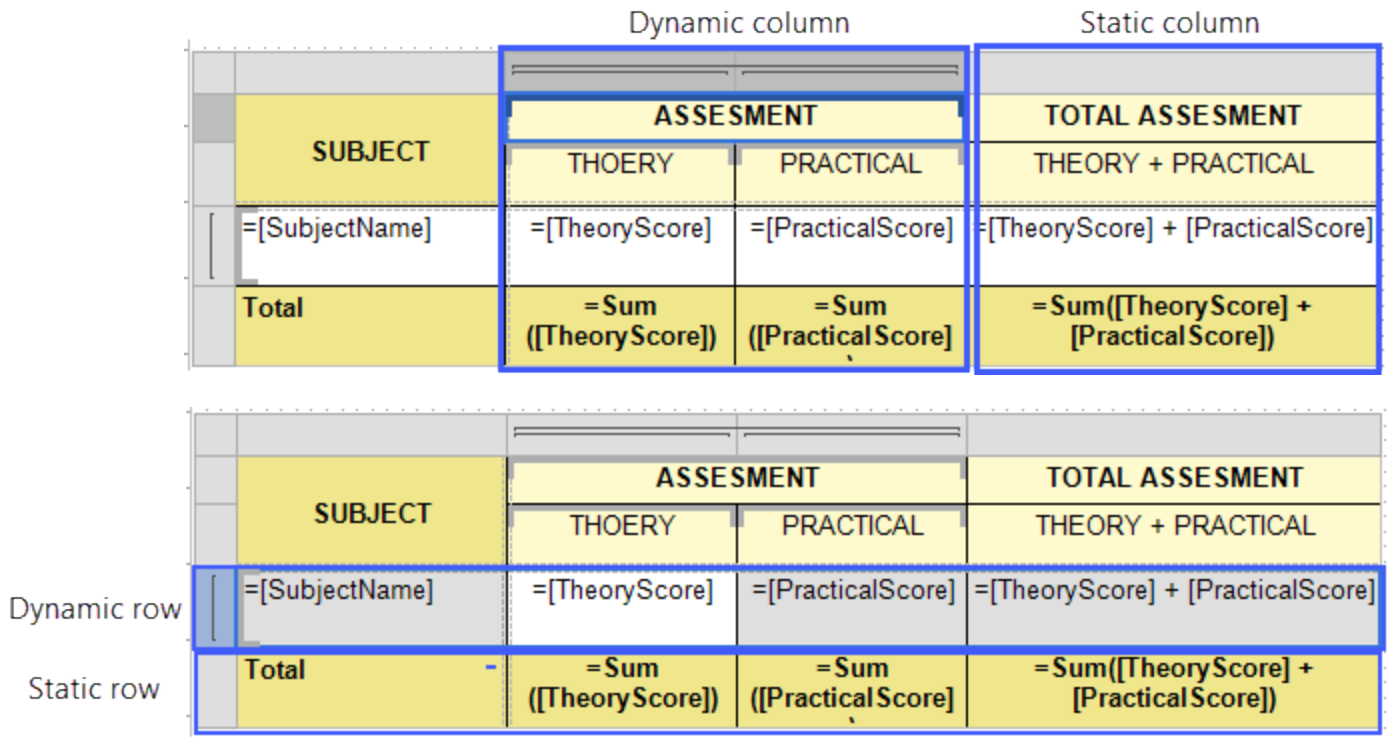
The image below displays the subjects in a row group. Nested column groups display practical and theory scores for the students. The total row displays the total scores for all of the subjects.

SUBJECT	ASSESSMENT		TOTAL ASSESSMENT
	THEORY	PRACTICAL	THEORY + PRACTICAL
Data Analysis	20	10	30
Software Testing	23	10	33
Distributed Systems	56	10	66
Operating Systems	45	10	55
Digital Systems	65	15	80
Total	**209**	**55**	**264**

To perform basic operations in the Tablix data region, we need to first understand the concept of static and dynamic rows and columns.

Rows or columns in the Tablix data region can be static or dynamic. The Tablix data region contains multiple rows and columns that provide a grid type layout, where you can add or remove static or dynamic rows and columns in order to display your data efficiently.

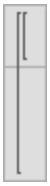
- **Static Rows and Columns** - A static row or column is not associated with any group data. When the report runs, a static row or column is rendered only once. Labels and totals are displayed using static rows or columns in Tablix data region.
- **Dynamic Rows and Columns** - A dynamic row or column is associated with one or more groups, and renders once for every unique value in the group. You can also create dynamic group rows or columns by adding a row group or a column group.



Row and Column Handles

When you select a Tablix data region, the row and column handles appear. These handles help you to work with the data region and visually specify the type of data added in your tablix layout.

The following table shows the different types of handles that appear in a Tablix data region.

Handle Icon	Description
	Row or column with one outer group.
	One outer group and one inner group.
	One outer group with an extra row for totals and one inner group.

Tablix Layout Actions

The Tablix data region provides context menu options to perform basic layout actions. You can access layout options for Tablix rows from the context menu by right-clicking on a selected row.

- **Insert Row:** Select from the following options to insert a row inside or outside of the selected group cell.

    - **Inside Group:** If a row group contains groups having distinct values, then as many rows are inserted as there are groups.
        - **Above**: Inserts a row above for each unique value of the row group.
        - **Below**: Inserts a row below for each unique value of the row group.
    - **Outside Group**: If a row group contains nested groups consisting of child and parent groups, then as many rows are inserted as there are parent groups.
        - **Above**: Inserts a row above for each unique value of the parent row group.
        - **Below**: Inserts a row below for each unique value of the parent row group.
- **Delete Row:** Delete the selected rows.
- **Distribute Rows Evenly**: Set the same height for multiple selected rows.
- **Add Row Group:** Select from the following options to insert row groups in a tablix.
    - **Parent Group**: To insert a parent row group.
    - **Child Group**: To insert a child row group.
    - **Adjacent Above**: To insert an adjacent row group above the selected row group.
    - **Adjacent Below**: To insert an adjacent row group below the selected row group.
- **Row Group:** Select the Delete Group option to delete a row group.

You can access layout options for Tablix columns from the context menu by right-clicking on a selected column.

- **Insert Column**: Select from the following options to insert a column inside or outside of the selected group cell.
    - **Inside Group**: If a column group contains groups having distinct values, then as many columns are inserted as there are groups.
        - **Left**: Inserts a column to the left for each unique value of the column group.
        - **Right**: Inserts a column to the right for each unique value of the column group.
    - **Outside Group**: If a column group contains nested groups consisting of child and parent groups, then as many columns are inserted as there are parent groups.
        - **Left**: Inserts a column to the left for each unique value of the parent column group.
        - **Right**: Inserts a column to the right for each unique value of the parent column group.
- **Delete Column**: Delete the selected columns.
- **Distribute Columns Evenly:** Set the same width for multiple selected columns.
- **Add Column Group:** Select from the following options to insert column groups in a tablix.
    - **Parent Group**: To insert a parent column group.
    - **Child Group**: To insert a child column group.
    - **Adjacent Left**: To insert an adjacent column group to the left of the selected column group.
    - **Adjacent Right**: To insert an adjacent column group to the right of the selected column group.
- **Column Group:** Select the Delete Group option to delete a column group.

Tablix Reports

Tablix data region can be used to display complex data using row groups and column groups. Let us look at a few scenarios to understand how Tablix data region works.

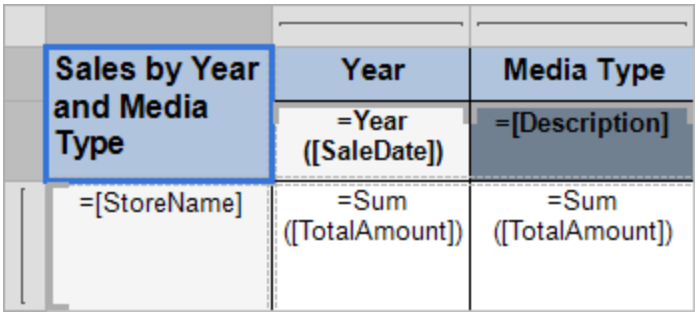
Using Multiple Adjacent Groups

In Tablix data region, unlike Matrix and Table data regions, it is possible to create multiple adjacent groups. Let us take an example of a Sales report to understand how grouping works in a Tablix data region.

Scenario

An organization wants to create a Sales report that displays sales data by year and media type. To design such a report

using Tablix data region, you need to create multiple adjacent groups in column group area, a single group in row group area, and display aggregated data in the body area.



Let us see how each area of Tablix data region works to create the desired output given below.

Column group area
The column group area contains two adjacent groups, namely **Year** and **Media Type**. The Year group displays the grouped data according to the year 2004 and 2005 and the Media Type group displays the grouped data values according to DVD, VHS, LaserDisc, HD-DVD media types stored in the database.

Row group area
The row group area contains a single group that is **StoreName**. Cells in the row group area display row group values and represent members of the row group hierarchy.

Body area
This area displays the aggregate sum of the **TotalAmount** in the Tablix. Cells in the tablix body area display detail data when the cells are in detail row or column and aggregated group data when the cell are in a group row or column.

Sales by Year and Media Type	Year		Media Type			
	2004	2005	DVD	VHS	LaserDisc	HD-DVD
Store #1006	$66,027.90	$88,526.10	$55,234.49	$57,072.52	$38,495.98	$3,751.01
Store #1003	$67,320.45	$120,158.53	$72,361.76	$60,231.29	$48,179.78	$6,706.15
Store #1002	$75,596.91	$124,393.77	$74,323.89	$69,476.62	$51,811.76	$4,378.42
Store #1001	$50,089.10	$102,818.20	$55,779.33	$49,959.71	$42,235.11	$4,933.16
Store #1005	$68,051.26	$90,181.60	$58,526.09	$55,561.34	$38,105.94	$6,039.48
Store #1004	$71,091.94	$80,189.28	$54,115.66	$55,000.11	$36,702.91	$5,462.54
Store #1000	$60,253.46	$100,999.49	$59,427.52	$55,805.88	$41,411.66	$4,607.88

Refer to Grouping in Tablix Walkthrough for detailed steps on how to create the above report using Tablix data region.

Using Cell Merging

In Tablix data region, it is possible to merge cells having duplicate values. Let us take an example of a Store Mangers report to understand how cell merging works in a Tablix data region.

Scenario

An organization wants to create a report to display names of all the store managers, using the concept of cell merging where cells with same value are merged automatically to avoid clutter.

To design such a report, you need to create nested groups in Tablix data region to display the name of managers according to the **Region**, **District** and **StoreName**.

Let us see how each area of Tablix data region works to create the desired output given below.

Column group area
The column group area contains no groups, however there are static labels for each column. There are four column labels, namely **Region**, **District**, **Store**, and **Manager** that are added as headers to describe information about the data.

Row group area
The row group area contains three groups that are nested in a parent/child relationship to display the row group data. The **Region** (parent) and **District** group (child) values are merged automatically to remove duplicate data values.

Body area
This area displays the full name of store managers. The body area displays the managers full name by concatenating two fields **FirstName** and **LastName** using the **&** operator in the expression.

Region	District	Store	Manager
No Region	No District	HQ	Lewis Bossert
Canada West	Vancouver	Store #1003	Wellington Crotto
		Store #1004	Wildon Caperton
		Store #1007	Liora Hyneman
	Victoria	Store #1001	Ginny Summer
		Store #1005	Alsy Strittmatter
North West	Portland	Store #1000	Aubri Moening
		Store #1006	Amber Jobin
	Seattle	Store #1002	Ernesto Mcdonell

Refer to Cell Merging in Tablix Walkthrough for detailed steps on how to create the above report using Tablix data region.

Data Sources and Datasets

Setting up a connection to the data source is the first step in binding data to the report. Once a connection is established, a data set is required to get the data you want to show on the report.

Data Sources

In ActiveReports, you can set the data source information in the Report Data Source Dialog.

The Report Data Source dialog is where you select the type of data to use, provide a connection string, and choose other options for your data source. You can also decide to use a shared data source, use a single transaction, and select a method for handling credentials. Once you add a data source, it appears in the Report Explorer under the Data Sources node. You can also add multiple data sources in a single report.

Datasets

A dataset fetches data from the data source to display in a report. The DataSet Dialog is where you provide a command type and query string and choose other options for your dataset.

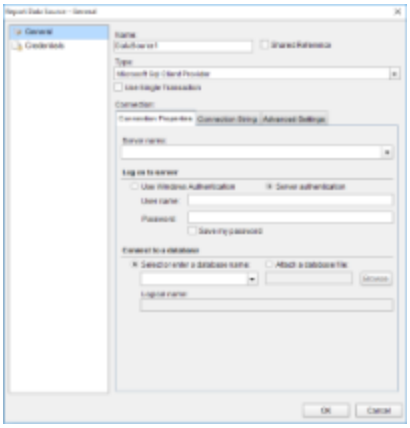
You can also control the timeout period and other data options, and add fields, parameters, and filters to fetch the data you need. With the XML data type, you have to add fields manually with XPath expressions. Once you have added a dataset, its fields appear under the Data Source node in the Report Explorer. You can add multiple datasets for a data source.

> **Note:** In both Page and RDL reports, you can also use multiple data sets, see Nested Data Regions Bound to Different Data for more information.

Report Data Source Dialog

You can access the Report Data Source dialog from the Report Explorer by doing one of the following:

- Click the **Add** icon on the top left and select **Data Source**.
- Right-click the Data Sources node and select **Add Data Source**.



The Report Data Source dialog provides the following pages where you can set data source properties:

General

The General page of the Report Data Source dialog is where you can set the Name, Type, and Connection string of a new data source, or choose to use a shared data source reference.

- In the **Name** field, you can enter a name for the data source. This name must be unique within the report. By default, the name is set to DataSource1.
- In the **Shared Reference** checkbox, you can select a shared data source reference. See Connect to a Data Source for further information. Once you have chosen the **Shared Reference** option, the **Reference** field to select a Shared Data Source becomes available. In the **Reference** field, select **From File** and select a shared data source file on your machine.
- If the **Shared Reference** checkbox is clear, you can select a data source type from the **Type** dropdown field. ActiveReports supports the following providers:
  - Microsoft SQL Client Provider
  - Csv Provider
  - DataSet Provider
  - JSON Provider
  - Microsoft ODBC Provider
  - Microsoft OleDb Provider
  - XML Provider
- You can also select to execute datasets that use this data source in a single transaction by checking the **Use Single Transaction** checkbox.
- If you select SQL, or OleDb as the data source **Type** value, the **Connection Properties**, **Connection String** and **Advanced Settings** pages appear under the **Connection** section. For XML data source, **Connection Properties** and **Connection String** pages appear. For JSON data source, **Schema**, **Content**, and **Connection String** pages appear. In other data source types, only the **Connection String** page appears.

Credentials

The Credentials page gives you the following four options for the **level of security** you need for the data in your report.

Use Windows Authentication

Select this option when you know that any users with a valid Windows account are cleared for access to the data, and you do not want to prompt them for a user name and password.

Use a specific user name and password

Select this option when you want to allow only a single user name and password to access the data in the report.

Prompt for credentials

Select this option when there is a subset of users who can access the data. The Prompt string textbox allows you to customize the text requesting a user name and password from users.

No credentials

Select this option only if the data in the report is for general public consumption.

Microsoft SQL Client Provider

The Microsoft SQL Client Data Provider supports following options under the Connection section in Report Data Source dialog.

Connection Properties

The Connection Properties tab gives access to properties specific to the following data types.

- **Server name**: This field requires you to enter a server name.
- **Log on to server**: Through this field, you can select whether to use Windows authentication or server authentication which requires a user name and password. Below this field you can also check the **Save my password** option for future reference.
- **Connect to a database**: Through this field, you can select whether to enter a database name or attach a database file.

Connection String

Sample Connection string

```
data source=in-data-sql\sql_2012;initial catalog=ADventurewORKS2012;user
id=user1;password=password@123
```

Advanced Settings

The Advanced Settings tab gives access to properties specific to each data type.

With the SQL data type, the Advanced Settings tab gives access to the following properties:

- **Application Name**: Indicates the client application name.
- **Auto Translate**: Indicates whether the OEM/ANSI characters are converted. You can set this property to True or False. By default, the value is set to True. If True then SQLOLEDB performs the OEM/ANSI character conversion when multi-byte character strings are retrieved from, or sent to, the SQL Server.
- **Current Language**: Indicates the SQL Server language name. It also identifies the language used for system

message selection and formatting. The language must be installed on the SQL Server, otherwise opening the connection will fail.
- **Network Address**: Indicates the network address of the SQL Server, specified by the Location property.
- **Network Library**: Indicates the name of the network library (DLL) used to communicate with the SQL Server. The name should not contain the path or the .dll file name extension. The default name is provided by the SQL Server client configuration.
- **Packet Size**: Indicates a network packet size in bytes. The Packet Size property value must be between 512 and 32767. By default, the SQLOLEDB network packet size is 4096.
- **Trusted Connection**: Indicates the user authentication mode. You can set this property to Yes or No. By default, the property value is set to No. If Yes, the SQLOLEDB uses the Microsoft Windows NT Authentication Mode to authorize user access to the SQL Server database, specified by the Location and Datasource property values. If this property is set to No, then the SQLOLEDB uses the Mixed mode to authorize user access to the SQL Server database. The SQL Server login and password are specified in the User Id and Password properties.
- **Use Procedure for Prepare**: Determines whether the SQL Server creates temporary stored procedures when Commands are prepared by the Prepared property.
- **Workstation ID**: Denotes a string that identifies the workstation.

CSV Provider

The CSV Data Provider supports following option under the Connection section in Report Data Source dialog.

Connection String

The CSV connection string is generated on the basis of options selected in the **Configure CSV Data Source** wizard.

Options in the Configure CSV Data Source wizard	Description	Example
Path	Path to the CSV file - both local and relative; or a URL for centrally located CSV data sources.	C:\Categories.csv
Encoding	Encoding of the CSV file.	Unicode (UTF-7)
File Type	The type of CSV file. You can choose from Fixed and Delimited options.	Delimited
Text Qualifiers	Symbol to specify where the text begins and ends. You can choose from Quotes and Single quotes options.	Quotes
Column Separator	Symbol to separate the columns. You can choose from Comma, Semicolon, Tab, and Space options.	Semicolon
Treat consecutive as one	Specify whether to join the column separators or row separators as one.	Checked for Column separator Unchecked for Row Separator
Locale	Specify the locale.	English (United States)
Starting Row	Row number to start fetching data.	0
Columns have	Specify whether the CSV file has columns with headers or not.	Checked

headers		
Row Separator	Symbol to separate the rows. You can choose from CRLF (carriage return and line feed), CR (carriage return), and LF (line feed) new line formats.	New line (CRLF)
Get from preview	Fills the Columns area with names and data types (string by default) for columns present in csv file. This allows you to modify the name and data type (as String, Boolean, Date Time, Integer, or Float) for the columns.	ClaimAmt(Float), SvcDate(DateTime)
Note: Text Qualifiers, **Column Separator, Row Separator**, and **Treat Consecutive as one** options are not available for **Fixed** file type.		

For example, the following connection string is generated based on the options selected in the Example column above.

```
Path=C:\\Categories.csv;Encoding=utf-7;Locale=en-
US;TextQualifier=";ColumnsSeparator=\;;RowsSeparator=\r\n;
Columns=EmployeeID,LastName,FirstName,Role,City;JoinColumnsSeparators=True;HasHeaders=True
```

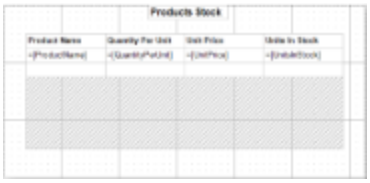
Reports with CSV Data

This topic explains the steps involved in connecting a page report to a CSV data source.

> **Note:**
>
> - This topic uses the **Products_header_tab.csv** sample database. The Products_header_tab.csv file can be downloaded from GitHub: ..\Samples14\Data\ Products_header_tab.csv.
> - Although this topic uses Page reports, you can also implement this using RDL reports.

When you complete these steps, the layout that looks similar to the following at design time and at run time.

Design-Time Layout



Run-Time Layout

Add an ActiveReport to the Visual Studio project

1. Create a new Visual Studio project.
2. In the **New Project** dialog that appears, select **ActiveReports 14 Page Report Application** and in the Name field, rename the file as **ProductsStock**.
3. Click **OK** to create a new **ActiveReports 14 Page Report Application**. By default a Page report is added to the project.

See Quick Start for information on adding different report layouts.

Connect the report to a CSV data source

1. In the Report Explorer, right-click the Data Sources node and select the **Add Data Source** option or select **Data Source** from the Add button.
2. In the **Report Data Source** dialog that appears, select the **General** page and enter the name of the data source. By default, the data source name is set to DataSource1. This name appears as a child node to the Data Sources node in the Report Explorer.
3. Under the **Type** field, select CSV Provider.
4. In the **Connection String** tab, click the **Build** icon to open the **Configure CSV Data Source** wizard.
5. To specify the **Path** of the file, click the **Open** button and navigate to [User folder]\Samples14\Data\Products_header_tab.csv file, which can be downloaded from GitHub.
6. Select the **Column Separator** as **Tab** from the drop-down menu. See the **Sample CSV Connection String** in the topic CSV Provider for further details.
7. Click **OK** to save the changes and close the **Configure CSV Data Source** wizard. The **Connection String** tab displays the generated connection string as shown below:
   ```
   Path=C:\\[User folder]\\Samples14\\Data
   \\Products_header_tab.csv;Locale=en-US;
   TextQualifier=";ColumnsSeparator= ;RowsSeparator=\r\n;HasHeaders=True
   ```
   You can validate the connection string by clicking the **Validate DataSource** icon.
8. Click **OK** to close the Report Data Source dialog. You have successfully connected the report to the CSV data source.

> **Note:** The dataset for a CSV data source is automatically added. The default name of the data set is name of the CSV file selected.

Add controls to the report

1. From the toolbox, drag a Table data region onto the design surface of the report.
2. Select the last column of the table, right-click, and select **Insert Column to the Right** option to add a fourth column.
3. Go to the Properties window and set the properties for the Table data region as described in the following section:

Property Name	Property Value
Location	0.375in, 0.5in
DataSetName	Products_header_tab
Size	5.5in, 0.75in
FixedSize	5.5in, 2in

4. From the Report Explorer, drag the fields into the Table Details row as described in the following section:

Field Name	TextBox

ProductName	TextBox4
QuantityPerUnit	TextBox5
UnitPrice	TextBox6
UnitsInStock	TextBox11

5. Select the Table Header row and set **FontWeight** to Bold.
6. From the toolbox, drag a TextBox onto the design surface of the report, go to the Properties window to set the properties as described in the following section:

Property Name	Property Value
Location	2.5in, 0in
Size	1.375in, 0.25in
Value	Products Stock
FontSize	12pt
FontWeight	Bold

View the report

- Click the preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

DataSet and Object Providers

The data source and data set for DataSet Provider and Object Provider data types can be set at run time. For more information, see Bind a Page Report to a Data Source at Run Time.

JSON Provider

The JSON Data Provider supports following options under the Connection section in Report Data Source dialog.

Content

In the Content tab, specify the type of JSON data source. The options available for specifying the JSON data are as follows:

- **External file or URL:** Enter the path or URL of an external JSON data file or select the file from the drop-down which displays the JSON files available in the same folder as the report. The connection string generated using this option starts with the keyword **jsondoc**.
- **Embedded:** Enter the path of the JSON data file to embed in the report. You can enter the data manually or edit the data in selected JSON file. The connection string generated using this option starts with the keyword **jsondata**.
- **Expression:** Enter an expression to bind to the JSON data at runtime. For more information on Expressions, see Use Dynamically Built JSON Data Source topic.

Schema

The JSON schema describes the structure of a JSON data. In ActiveReports, the JSON data provider uses the JSON schema to obtain fields. For more information on JSON schema, please see http://json-schema.org/draft/2019-09/json-schema-core.html.
The keywords of JSON schema that are supported in the JSON data provider are:

- **type**: Indicates the type of the JSON schema element. See here for more information on type keyword.
- **properties**: Indicates the properties collection for JSON schema elements with the object type. See here for more information on properties keyword.
- **items**: Indicates the definition of items for JSON schema elements with array type. Only single values are supported.
  For example, `"items" : [ {...}, {...}, {...} ]` is not supported because it contains multiple values.
  See here for more information on items keyword.
- **definitions**: Indicates the independent definitions which can be used by other JSON schema elements using `$ref` keyword. See here for more information on definitions keyword.
- $**ref**: Indicates the reference to a definition for JSON schema elements with object type. Only "definitions" (`{ $ref : #/definitions/... }`) references are supported.

> **Note:** Schema is the only required option to create a connection string.

In the **Schema** tab, the options available for specifying the JSON schema are:

- **Auto:** This is the default option that auto generates the schema.
- **External file or URL:** Enter the path or URL of an external JSON schema file or select the file from the drop-down which displays the JSON files available in the same folder as the report is located. The connection string generated using this option starts with the keyword **schemadoc**.
- **Embedded:** Enter the path of the JSON schema file to embed in the report. You can enter the schema manually or edit the schema in the selected JSON file. The connection string generated using this option starts with the keyword **schemadata**.

  For generating JSON schema, use the JSON schema generator available at http://jsonschema.net/.

Connection String

JSON connection string has two parts - **jsondoc** or **jsondata** and schemadoc or **schemadata**.

- **jsondoc or jsondata:** Refers to a specific JSON data file located on either the file system or at a web-accessible location.
- **schemadoc or schemadata:** Refers to JSON schema file corresponding to the existing JSON data.

  For example,

```
jsondoc=C:\Data\customers.json;schemadata={
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "address": {
```

```
        "type": "object",
        "properties": {
          "streetAddress": {
            "type": "string"
          },
          "city": {
            "type": "string"
          }
        },
        "required": [
          "streetAddress",
          "city"
        ]
      },
      "phoneNumber": {
        "type": "array",
        "items": {
          "type": "object",
          "properties": {
            "location": {
              "type": "string"
            },
            "code": {
              "type": "integer"
            }
          },
          "required": [
            "location",
            "code"
          ]
        }
      }
    },
    "required": [
      "address",
      "phoneNumber"
    ]
  }
```

Note: If you are using an expression in the connection string, you should use single quotes (') instead of double quotes (") in jsondoc or jsondata and schemadoc or schemadata.

For example, the following connnection string is invalid:

```
="jsondata={  "Name": "Name"};schemadata={    "$schema": "http://json-
schema.org/draft-04/schema#",
    "definitions": {},    "id": "http://example.com/example.json",
    "properties": {        "Name": {            "id": "/properties/Name",
```

```
            "type": "string"        }    },    "type": "object"}"
```

Use the following instead:

```
="jsondata={  'Name': 'Name'};schemadata={    '$schema': 'http://json-
schema.org/draft-04/schema#',
    'definitions': {},    'id': 'http://example.com/example.json',
    'properties': {        'Name': {            'id': '/properties/Name',
            'type': 'string'        }    },    'type': 'object'}"
```

Reports with JSON Data

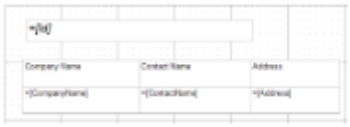
This topic explains the steps involved in connecting a page report to a JSON data source at run time and using a web service to fetch the data with the authorized access.

Note:

- You must have the IIS Express installed on your machine.
- This topic uses the **Customers** sample data file. The customers.json file can be downloaded from GitHub: ..\Samples14\Data\customers.json.

When you complete these steps, you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout



Run-Time Layout

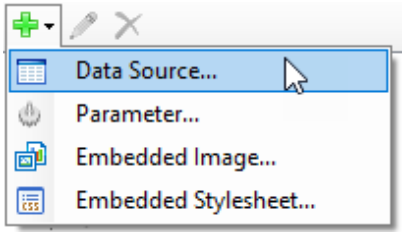
Add an ActiveReport to the Visual Studio project

1. Create a new Visual Studio project.
2. In the **New Project** dialog that appears, select **ActiveReports 14 Page Report Application** and in the Name field, rename the file as **CustomerDetails**.
3. Click **OK** to create a new **ActiveReports 14 Page Report Application**. By default a Page report is added to the project.
4. Right-click the solution and select **Restore Nuget Package**.

See Quick Start for information on adding different report layouts.

Connect the report to a data source

1. In the Report Explorer, right-click the Data Sources node and select the **Add Data Source** option or select **Data Source** from the Add button.



2. In the Report Data Source Dialog that appears, select the **General** page and enter the name of the data source.
3. Under the **Type** field, select **Json Provider**.
4. On the same page, select **Embedded** under Schema and enter the following schema to be embedded.

JSON schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "Customers": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "Id": {
            "type": "string"
          },
          "CompanyName": {
            "type": "string"
          },
          "ContactName": {
            "type": "string"
          },
          "ContactTitle": {
            "type": "string"
          },
          "Address": {
            "type": "string"
          },
          "City": {
            "type": "string"
          },
          "PostalCode": {
            "type": "string"
          },
          "Country": {
            "type": "string"
          },
          "Phone": {
```

```
            "type": "string"
          },
          "Fax": {
            "type": "string"
          }
        },
        "required": [
          "Id",
          "CompanyName",
          "ContactName",
          "ContactTitle",
          "Address",
          "City",
          "PostalCode",
          "Country",
          "Phone",
          "Fax"
        ]
      }
    },
    "ResponseStatus": {
      "type": "object",
      "properties": {}
    }
  },
  "required": [
    "Customers",
    "ResponseStatus"
  ]
}
```

Note: The schema shown above has been generated for **customers.json** data, using the JSON schema generator available at http://jsonschema.net.

5. Click **OK** to save the data source connection.

Add a dataset

1. In the Report Explorer, right-click the data source node and select the **Add Data Set** option or select **Data Set** from the Add button.
2. In the DataSet Dialog that appears, select the **General** page and enter the name of the dataset as **Customers**.
3. On the **Query** page, click **Edit with JSON Query Designer** button and choose the JSONPath upto `[*]` to obtain the following query in the **Query** text box: `$.Customers[*]`
4. Click **OK** to close the dialog. Your dataset and queried fields appear as nodes in the Report Explorer.

Add controls to the report

1. From the Report Explorer, drag the **Id** field onto the design surface of the report and go to the Properties window to set the properties as follows.

Property Name	Property Value

Location	0.375in, 0.25in
FontSize	14pt
FontStyle	Italic
Size	4in, 0.375in

2. From the Report Explorer, drag the Table data region onto the design area and go to the Properties window to set the properties as follows.

Property Name	Property Value
Location	0.375in, 1in
Size	4.5in, 1.125in
FixedSize	4.5in, 0.75in

3. Right-click any row handle to the left of the table and click **Table Footer** to remove the table footer row.
4. Hover over TextBox5 to reveal the field selection adorner, click it to display a list of available fields, and select the **CompanyName** field. This automatically adds a static label in the table header row.
5. Hover over TextBox6 to reveal the field selection adorner, click it to display a list of available fields, and select the **ContactName** field. This automatically adds static label in the table header row.
6. Hover over TextBox7 to reveal the field selection adorner, click it to display a list of available fields, and select the **Address** field. This automatically adds static label in the table header row.

Display a report in the Viewer

1. From Solution Explorer, open Form1.
2. From the Visual Studio toolbox, drag the Viewer control onto the Form1.
3. Set the viewer's **Dock** property to **Fill** to show the complete Viewer control on the Form1 and set the viewer's **Name** property to **reportPreview**.
4. Double-click the title bar of the Form1 to create an event-handling method for the **Form1_Load** event.
5. In Form1.cs, paste the following code after InitializeComponent method to load data in the report and display the report in the viewer.

Form1.cs

```
// The handler of <see cref="PageDocument.LocateDataSource"/> that returns
appropriate data for a report.
        private void OnLocateDataSource(object sender,
GrapeCity.ActiveReports.LocateDataSourceEventArgs args)
        {
            object data = null;
            var dataSourceName = args.DataSourceName;
            var source = new DataLayer();
            if (dataSourceName == "DataSource1")
            {
                data = source.CreateData();
            }
            args.Data = data;
        }
// Loads and shows the report.
        private void Form1_Load(object sender, EventArgs e)
        {
```

```
            var rptPath = new System.IO.FileInfo(@"..\..\Customers.rdlx");
            var definition = new GrapeCity.ActiveReports.PageReport(rptPath);
            definition.Document.LocateDataSource += OnLocateDataSource;
            reportPreview.ReportViewer.LoadDocument(definition.Document);
        }
```

Add a Web service project

The web service added to the project authenticates access to the JSON data.

1. From the **File** menu, go to **Add**, and then select **New Project**.
2. In the New Project dialog that appears, select **ASP.NET Web Application** and in the Name field, rename the file as **WebService**.
3. Click OK to add the new project.
4. In the **New ASP.NET Project** dialog, select **Empty** template and click OK.
5. In the Solution Explorer, right-click the **WebService** node, go to **Add**, and select **Add Class**.
6. In the Add New Item dialog that appears, select **Class** and in the Name field, rename the file as **BasicAuthHttpModule.cs.**
7. Click the **Add** button to add the new class to the project.
8. In the BasicAuthHttpModule class file that opens, add the following code inside the WebService namespace.

BasicAuthHttpModule.cs

```
namespace WebService
{
    public class BasicAuthHttpModule : IHttpModule
    {
        private const string Realm = "My Realm";
        public void Init(HttpApplication context)
        {
            // Register event handlers
            context.AuthenticateRequest += OnApplicationAuthenticateRequest;
            context.EndRequest += OnApplicationEndRequest;
        }
        private static void SetPrincipal(System.Security.Principal.IPrincipal
principal)
        {
            System.Threading.Thread.CurrentPrincipal = principal;
            if (HttpContext.Current != null)
            {
                HttpContext.Current.User = principal;
            }
        }
        // Validate the username and password.
        private static bool CheckPassword(string username, string password)
        {
            return username == "admin" && password == "1";
        }
        private static void AuthenticateUser(string credentials)
        {
            try
```

```
            {
                var encoding = System.Text.Encoding.GetEncoding("iso-8859-1");
                credentials =
encoding.GetString(Convert.FromBase64String(credentials));
                int separator = credentials.IndexOf(':');
                string name = credentials.Substring(0, separator);
                string password = credentials.Substring(separator + 1);
                if (CheckPassword(name, password))
                {
                    var identity = new
System.Security.Principal.GenericIdentity(name);
                    SetPrincipal(new
System.Security.Principal.GenericPrincipal(identity, null));
                }
                else
                {
                    // Invalid username or password.
                    HttpContext.Current.Response.StatusCode = 403;
                }
            }
            catch (FormatException)
            {
                // Credentials were not formatted correctly.
                HttpContext.Current.Response.StatusCode = 401;
            }
        }
        private static void OnApplicationAuthenticateRequest(object sender, EventArgs
e)
        {
            var request = HttpContext.Current.Request;
            System.IO.FileInfo info = new
System.IO.FileInfo(request.Url.AbsolutePath);
            if (!info.Name.Equals("GetJson")) return;
            var authHeader = request.Headers["Authorization"];
            if (authHeader != null)
            {
                var authHeaderVal =
System.Net.Http.Headers.AuthenticationHeaderValue.Parse(authHeader);
                // RFC 2617 sec 1.2, "scheme" name is case-insensitive
                if (authHeaderVal.Scheme.Equals("basic",
                        StringComparison.OrdinalIgnoreCase) &&
                    authHeaderVal.Parameter != null)
                {
                    AuthenticateUser(authHeaderVal.Parameter);
                }
            }
            else
            {
                HttpContext.Current.Response.StatusCode = 401;
```

```
            }
        }
        // If the request was unauthorized, add the WWW-Authenticate header
        // to the response.
        private static void OnApplicationEndRequest(object sender, EventArgs e)
        {
            var response = HttpContext.Current.Response;
            if (response.StatusCode == 401)
            {
                response.Headers.Add("WWW-Authenticate",
                        string.Format("Basic realm=\"{0}\"", Realm));
            }
        }
        public void Dispose()
        {
        }
    }
}
```

9. Add reference to **System.Net.Http.dll** in the WebService project.
10. In the Solution Explorer, right-click the **WebService** node, go to **Add**, and select **New Item**.
11. In the Add New Item dialog that appears, select **Web Service** and in the Name field, rename the file as **Service.asmx**.
12. Click the **Add** button to add the new service to the project.
13. In the Service file that opens, uncomment `[System.Web.Script.Services.ScriptService]` line of code and add the following code below the WebMethod.

Service.asmx.cs

```
[System.Web.Script.Services.ScriptMethod(UseHttpGet = true, ResponseFormat =
System.Web.Script.Services.ResponseFormat.Json)]
        public string GetJson()
        {
            string result;
            try
            {
                using (System.IO.StreamReader streamReader = new
System.IO.StreamReader(Properties.Resource.JsonFilePath, System.Text.Encoding.UTF8))
                {
                    result = streamReader.ReadToEnd();
                }
            }
            catch (System.IO.FileNotFoundException e)
            {
                var errorMessage =
String.Format(Properties.Resource.FormatErrorMessage, e.Message, e.StackTrace);
                result = "{'error': '" + errorMessage + "'}";
            }
            return result;
        }
```

14. In the Solution Explorer, right-click the **WebService** node, go to **Add**, and select **New Item**.
15. In the Add New Item dialog that appears, select **Web Form** and in the Name field, rename the file as **default.aspx**.

16. Click the **Add** button to add a new file to the project.
17. In the default.aspx code file that opens, replace tags with <form> tags as shown.

default.aspx

```
<form id="form1" runat="server">
    <asp:Label ID="messageLabel" runat="server" Text=""></asp:Label>
</form>
```

18. In Solution Explorer, right-click default.aspx and select **View Code**.
19. In the default.aspx code file that opens, add the following code to the Page_Load event.

default.aspx.cs

```
messageLabel.Text = Properties.Resource.bodyOfMessage;
```

20. In the Solution Explorer, right-click the **WebService** node, go to **Add**, and select **New Item**.
21. In the Add New Item dialog that appears, select **Resources File** and in the Name field, rename the file as **Resource.resx**.
22. Click the **Add** button to add a new file to the project. Make sure that the Resources file is located in the **WebService>Properties** node.
23. Add the following data to the Resource file.

Name	Value
bodyOfMessage	Json Data Source Sample WebService was started successfully
FormatErrorMessage	Message : {0}, StackTrace: {1}
JsonFilePath	[User folder]\Samples14\Data\customers.json

24. In the Solution Explorer, right-click the solution node and select **Properties**.
25. In the **Solution Property Pages** dialog that appears, select **Multiple startup projects** and then the **Start** action for each of the two projects.
26. Click **Apply** and then **OK** to apply the changes to the solution.

Add the DataLayer class

The DataLayer class provides the data used in the walkthrough.

1. Right-click **WebService** project, go to **Add** and select **Class**.
2. In the Add New Item dialog that appears, select **Class** and in the Name field, rename the file as **DataLayer.cs**.
3. Click the **Add** button to add the new class to the project.
4. In the DataLayer class file that opens, add the following code inside the project namespace.

Caution: It is required to change the URL port number set in "source_url" (the part after http://localhost:) to the port number that the IIS Express is actually using. The port number used by IIS Express can be confirmed or changed on the property page of the WebService project.

DataLayer.cs

```
// Provides the data used in the sample.
 internal sealed class DataLayer
```

```
    {
        public String CreateData()
        {
            string source_url = @"http://localhost:6719/Service.asmx/GetJson";
            string responseText = null;
            using (var webClient = new System.Net.WebClient())
            {
                webClient.Headers[System.Net.HttpRequestHeader.Authorization] =
"Basic " + Convert.ToBase64String(System.Text.Encoding.Default.GetBytes("admin:1"));
// username:password
                webClient.Headers[System.Net.HttpRequestHeader.ContentType] =
"application/json;";
                webClient.Encoding = System.Text.Encoding.UTF8;
                var responseJson = webClient.DownloadString(source_url);
                Dictionary<string, string> values = new
System.Web.Script.Serialization.JavaScriptSerializer().Deserialize<Dictionary<string,
string>>(responseJson);
                if (values.ContainsKey("d"))
                {
                    responseText = values["d"];
                }
            }
            return responseText;
        }
    }
```

Modify the Web.config file

1. From the Solution Explorer, open the Web.config file inside the WebService project.
2. Replace the <configuration> tags with following code.

Web.config

```
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.6.2"/>
    <httpRuntime targetFramework="4.6.2"/>
    <httpModules>
      <add name="ApplicationInsightsWebTracking"
type="Microsoft.ApplicationInsights.Web.ApplicationInsightsHttpModule,
Microsoft.AI.Web"/>
    </httpModules>
  <webServices>
      <protocols>
        <add name="HttpGet"/>
        <add name="HttpPost"/>
        <add name="HttpSoap"/>
      </protocols>
</webServices>
```

```
  </system.web>
  <system.codedom>
    <compilers>
      <compiler language="c#;cs;csharp" extension=".cs"
        type="Microsoft.CodeDom.Providers.DotNetCompilerPlatform.CSharpCodeProvider,
Microsoft.CodeDom.Providers.DotNetCompilerPlatform, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"
        warningLevel="4" compilerOptions="/langversion:6 /nowarn:1659;1699;1701"/>
      <compiler language="vb;vbs;visualbasic;vbscript" extension=".vb"
        type="Microsoft.CodeDom.Providers.DotNetCompilerPlatform.VBCodeProvider,
Microsoft.CodeDom.Providers.DotNetCompilerPlatform, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"
        warningLevel="4" compilerOptions="/langversion:14 /nowarn:41008
/define:_MYTYPE=\&quot;Web\&quot; /optionInfer+"/>
    </compilers>
  </system.codedom>
  <system.webServer>
    <validation validateIntegratedModeConfiguration="false"/>
    <modules>
      <add name="BasicAuthHttpModule" type="WebService.BasicAuthHttpModule,
WebService"/>
      <remove name="ApplicationInsightsWebTracking"/>
      <add name="ApplicationInsightsWebTracking"
type="Microsoft.ApplicationInsights.Web.ApplicationInsightsHttpModule,
Microsoft.AI.Web"
        preCondition="managedHandler"/>
    </modules>
  </system.webServer>
</configuration>.
```

View the report

1. From the Build menu option, select **Build Solution**.
2. Press **F5** to run the project.

Microsoft ODBC Provider

The Microsoft ODBC Data Provider supports following option under the Connection section in Report Data Source dialog.

Connection String

Sample Odbc Connection String

```
Driver=Microsoft Access Driver (*.mdb);Dbq=C:\nwind.mdb;
```

Microsoft OLeDb Provider

The Microsoft OleDb Data Provider supports following options under the Connection section in Report Data Source

dialog.

Connection Properties

The Connection Properties tab gives access to properties specific to the following data types.

- **OLE DB Provider**: This field requires you to select one among the list of OLE DB Providers provided in the drop down list. You need to be selective in the choice of the provider. For example, Microsoft.Jet.OLEDB.4.0 is not supported in 64 bit OS while Microsoft.ACE.OLEDB.12.0 is supported. See Troubleshooting article if an exception occurs on previewing reports connecting Microsoft Access OLE DB provider in a 64-bit system.
- **Enter a server or file name**: This field requires you to enter a server or a file name along with its location.
- **Log on to server**: Through this field you can select whether to use Windows NT integrated security or use a specific user name and password.

Connection String

Sample OleDb Connection String

```
provider=Microsoft.Jet.OLEDB.4.0;data source=c:\nwind.mdb;
```

Advanced Settings

With the OleDb data type, the Advanced Settings tab gives access to the Microsoft Jet OLEDB provider-specific connection parameters.

- **Jet OLEDB: Compact Reclaimed Space Amount**: Indicates an estimate of the amount of space, in bytes, that can be reclaimed by compacting the database. This value is only valid after a database connection has been established.
- **Jet OLEDB: Connection Control**: Indicates whether users can connect to the database.
- **Jet OLEDB: Create System Database**: Indicates whether to create a system database when creating a new data source.
- **Jet OLEDB: Database Locking Mode**: Indicates the locking mode for this database. The first user to open the database determines what mode to use when the database is open.
- **Jet OLEDB: Database Password**: Indicates the database password.
- **Jet OLEDB: Don't Copy Locale on Compact**: Indicates whether the Jet should copy locale information when compacting a database.
- **Jet OLEDB: Encrypt Database**: Indicates whether a compacted database should be encrypted. If this property is not set, the compacted database will be encrypted if the original database was encrypted.
- **Jet OLEDB: Engine Type**: Indicates the storage engine to access the current data store.
- **Jet OLEDB: Exclusive Async Delay**: Indicates the maximum length of time, in milliseconds, that the Jet can delay asynchronous writes to disk when the database is opened exclusively. This property is ignored unless Jet OLEDB: Flush Transaction Timeout is set to 0.
- **Jet OLEDB: Flush Transaction Timeout**: Indicates the amount of time before data stored in a cache for asynchronous writing is actually written to disk. This setting overrides the values for Jet OLEDB:Shared Async Delay and jet OLEDB: Exclusive Async Delay.
- **Jet OLEDB: Global Bulk Transactions**: Indicates whether the SQL bulk transactions are transacted.
- **Jet OLEDB: Global Partial Bulk Ops**: Indicates the password to open the database.
- **Jet OLEDB: Implicit Commit Sync**: Indicates whether the changes made in internal implicit transactions are written in synchronous or asynchronous mode.
- **Jet OLEDB: Lock Delay**: Indicates the number of milliseconds before attempting to acquire a lock after a previous attempt has failed.

- **Jet OLEDB: Lock Retry**: Indicates the frequency of attempts to access a locked page.
- **Jet OLEDB: Max Buffer Size**: Indicates the maximum amount memory, in kilobytes, the Jet can use before it starts flushing changes to disk.
- **Jet OLEDB: MaxLocksPerFile**: Indicates the maximum number of locks the Jet can place on a database. The default value is 9500.
- **Jet OLEDB: New Database Password**: Indicates the new password for this database. The old password is stored in Jet OLEDB: Database Password.
- **Jet OLEDB: ODBC Command Time Out**: Indicates the number of milliseconds before a remote ODBC query from the Jet will timeout.
- **Jet OLEDB: Page Locks to Table Lock**: Indicates how many pages to lock within a transaction before the Jet attempts to promote the lock to a table lock. If this value is 0, then the lock is never promoted.
- **Jet OLEDB: Page Timeout**: Indicates the number of milliseconds before the Jet will check if its cache is out of date with the database file.
- **Jet OLEDB: Recycle Long-Valued Pages**: Indicates whether the Jet should aggressively try to reclaim BLOB pages when they are freed.
- **Jet OLEDB: Registry Path**: Indicates the Windows registry key that contains values for the Jet database engine.
- **Jet OLEDB: Reset ISAM Stats**: Indicates whether the schema Recordset DBSCHEMA_JETOLEDB_ISAMSTATS should reset its performance counters after returning performance information.
- **Jet OLEDB: Shared Async Delay**: Indicates the maximum amount of time, in milliseconds, the Jet can delay asynchronous writes to disk when the database is opened in the multi-user mode.
- **Jet OLEDB: System Database**: Indicates the path and file name for the workgroup information file (system database).
- **Jet OLEDB: Transaction Commit Mode**: Indicates whether the Jet writes data to disk synchronously or asynchronously when a transaction is committed.
- **Jet OLEDB: User Commit Sync**: Indicates whether changes made in transactions are written in the synchronous or the asynchronous mode.

XML Provider

The XML Data Provider supports following options under the Connection section in Report Data Source dialog.

Connection Properties

- **External file or URL:** This field requires you to enter the path of an external XML source such as a local file or the http location of a file.
- **Embedded:** This field requires you to enter the path of the XML file to embed in the report. You can also enter the data manually or edit the data in selected XML file.
- **Expression:** This field requires you to enter the path expression. User can also enter the path expression in the Connection String.

Connection String

- **xmldoc:** Refers to a specific XML file located on either the file system or at a web-accessible location. For example, `xmldoc=C:\MyXmlFile.xml;`
- **xmldata:** Provides specific XML data in the Connection String itself. For example,

```
xmldata=<people>
  <person>
    <name>
      <given>John</given>
```

```
      <family>Doe</family>
    </name>
  </person>
  <person>
    <name>
      <given>Jane</given>
      <family>Smith</family>
    </name>
  </person>
</people>;
```

- **TransformationDoc:** Refers to a specific XSLT file to apply to the XML data.

Note that elements in the Connection String must be terminated with a semicolon (;) character.

Reports with XML Data

This topic explains the steps involved in connecting a page report to an XML data source and creating a dataset. It also demonstrates the use of the List control.

> **Note:**
>
> - This topic uses the **Factbook** sample database. The Factbook.xml file can be downloaded from GitHub: ..\Samples14\Data\Factbook.xml.
> - Although this topic uses Page reports, you can also implement this using RDL reports.

When you complete these steps you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout

Run-Time Layout

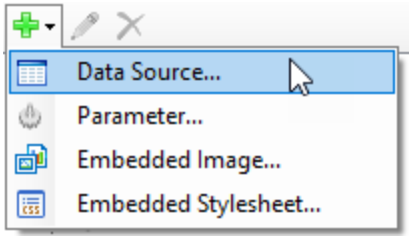
Add an ActiveReport to the Visual Studio project

1. Create a new Visual Studio project.
2. In the **New Project** dialog that appears, select **ActiveReports 14 Page Report Application** and in the Name field, rename the file as **ExchangeRates**.
3. Click **OK** to create a new **ActiveReports 14 Page Report Application**. By default a Page report is added to the project.

See Quick Start for information on adding different report layouts.

Connect the report to a data source

1. In the Report Explorer, right-click the Data Sources node and select the **Add Data Source** option or select **Data Source** from the Add button.



2. In the Report Data Source Dialog that appears, select the **General** page and in the Name field, enter a name like **Factbook**.
3. Under the **Type** field, select XML Provider.
4. In the **Connection Properties** tab, select the type of XML data as External file or URL.
5. Click the dropdown in **Select or type the file name or URL** field.
6. Select **<Browse...>** navigate to Factbook.xml. See Connect to a Data Source for information on connecting to a data source.
7. Click the **Connection String** tab. The connection string that gets generated is `xmldoc=[User Folder]\Samples14\Data\Factbook.xml`. You can validate the connection string by clicking the **Validate DataSource** icon .
8. Click **OK**.

Add a dataset

1. In the Report Explorer, right-click the data source node and select the **Add Data Set** option or select **Data Set** from the Add button.
2. In the DataSet Dialog that appears, select the **General** page and enter the name of the dataset as **ExchangeRates**.
3. On the **Query** page of this dialog, click the **Edit with XML Query Designer** icon to open **XML DataSet Query Builder** dialog.
4. Choose the XPath upto **Country** from the tree nodes. The following XML query is generated:
   `countries/country`
5. Click the **Validate DataSet** icon at the top right hand corner above the Query box to validate the query. You can see all the fields available in the dataset on the **Fields** page.
6. Click **OK** to close the dialog. Your data set and queried fields appear as nodes to the data source in the Report Explorer.

Add controls to the report

1. From the toolbox, drag a List data region onto the design surface of the report and go to the Properties window to set the **DataSetName** property to ExchangeRates.
2. From the Report Explorer, drag the **@name** field onto the list, center it at the top, and go to the Properties window

to set the **FontSize** property to 14pt.

3. From the Report Explorer, drag the following fields onto the list with properties set as described in the table below.

Field Name	Property Name
Currency	Location: 1.125in, 0.5in Size: 2.25in, 0.25in
VsUSD2004	Location: 4.5in, 0.875in Size: 1in, 0.25in
VsUSD2003	Location: 4.5in, 1.25in Size: 1in, 0.25in
VsUSD2002	Location: 4.5in, 1.625in Size: 1in, 0.25in
VsUSD2001	Location: 4.5in, 2in Size: 1in, 0.25in
VsUSD2000	Location: 4.5in, 2.375in Size: 1in, 0.25in

> **Note**: You will notice that the expressions created for these fields are different than usual. Because Visual Basic syntax does not allow an identifier that begins with a number, any numeric field names must be treated as strings in expressions.

4. From the toolbox, drag a TextBox onto the list and go to the Properties window to set the properties as described in the table below to combine static text with a field value.

Property Name	Property Value
Location	0.145in, 0.875in
Size	3in, 0.25in
Value	="Value of " & Fields!Currency.Value & " versus US$ for year:"

5. From the toolbox, drag TextBox controls onto the list and go to the Properties window to set the properties as described in the table below to create static labels.

TextBox1

Property Name	Property Value
Location	0.125in, 0.5in
Size	0.75in, 0.25in
FontWeight	Bold
Value	Currency:

TextBox2

Property Name	Property Value

Location	3.375in, 0.875in
Size	1in, 0.25in
TextAlign	Right
Value	2004:

TextBox3

Property Name	Property Value
Location	3.375in, 1.25in
Size	1in, 0.25in
TextAlign	Right
Value	2003:

TextBox4

Property Name	Property Value
Location	3.375in, 1.625in
Size	1in, 0.25in
TextAlign	Right
Value	2002:

TextBox5

Property Name	Property Value
Location	3.375in, 2in
Size	1in, 0.25in
TextAlign	Right
Value	2001:

TextBox6

Property Name	Property Value
Location	3.375in, 2.375in
Size	1in, 0.25in
TextAlign	Right
Value	2000:

View the report

- Click the preview tab to view the report at design time.

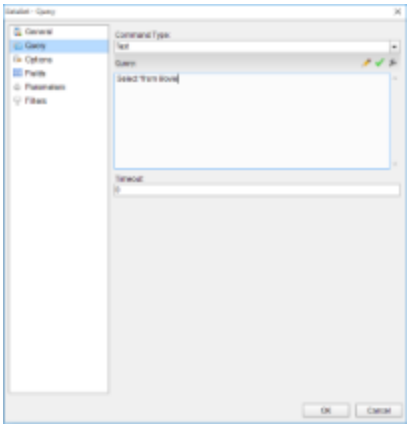
OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

DataSet Dialog

You can access the DataSet dialog from the Report Explorer by doing one of the following:

- With the Data Source node (like DataSource1) selected, click the **Add** icon on the top left and select **Data Set**.
- Right-click an existing data source and select **Add Data Set**.



The DataSet dialog provides the following pages where you can set dataset properties:

General

The General page of the DataSet dialog is where you can set the Name of the dataset.

Name: In the **Name** field, you can enter a name for the dataset. By default, the name is set to DataSet1. The name of the dataset appears in the tree view of the Report Explorer. It is also used to call the dataset in code so it should be unique within the report.

Query

The Query page of the DataSet dialog is where you set the SQL query, stored procedure or table to define the data you want to fetch in the dataset of your report.

Command type: You can choose from the three enumerated command types.

Type	Description
Text	Choose Text if you want to write a SQL query to retrieve data.
StoredProcedure	Choose StoredProcedure if you want to use a stored procedure.
TableDirect	Choose TableDirect if you want to return all rows and columns from one or more tables.

Query: Based on the command type you select above, you can set the query string in this field.

> **Note:**

- If you select the TableDirect command type, you may need to use escape characters or qualifying characters in case any of the table names include special characters.
- Specify the calculated index for arrays in a JSONPath expression in the following ways:
  - To obtain the last entry in an array, use `-1:` in square brackets. For example, use `$..book[-1:]`.
  - To obtain evaluated expressions correctly, the field names in square brackets should be in single quotes. For example, use `$..book[0]['category','author']`.

To create multiple datasets based on the JSON data provider, check **Select multiple nodes** option in the **JSON Query Builder**.

Timeout: You can set the number of seconds that you want the report server to wait for the query to return the data before it stops trying.

Options

The Options page is where you select one of the various options available to the dataset.

CaseSensitivity: Set this value to Auto, True, or False to indicate whether to make distinctions between upper and lower case letters. Auto, the default value, causes the report server to get the value from the data provider. If the data provider does not set the value, the report runs without case sensitivity.

Collation: Choose from Default or a country from the list to indicate which collation sequence to use to sort data. The Default value causes the report server to get the value from the data provider. If the data provider does not set the value, the report uses the server locale. This is important with international data, as the sort order for different languages can be different from the machine sort.

KanaTypeSensitivity: Set this value to Auto, True, or False with Japanese data to indicate whether distinctions are made between Hiragana and Katakana kana types. Auto, the default value, causes the report server to get the value from the data provider. If the data provider does not set the value, the report runs without kana type sensitivity.

WidthSensitivity: Set this value to Auto, True, or False with Japanese data to indicate whether distinctions are made between single-byte (half-width) characters and double-byte (full-width) characters. Auto, the default value, causes the report server to get the value from the data provider. If the data provider does not set the value, the report runs without width sensitivity.

AccentSensitivity: Set this value to Auto, True, or False to indicate whether distinctions are made between accented and unaccented letters. Auto, the default value, causes the report server to get the value from the data provider. If the data provider does not set the value, the report runs without accent sensitivity.

Fields

The **Fields** page of the DataSet dialog populates automatically for OleDb, ODBC, SQL, JSON, and XML data providers. To see a list of fields in the **Name** and **Value** columns of the Fields page, enter a valid query, table name, or stored procedure on the Query page.

Note: The dataset for a CSV data source is automatically created on adding the data source. You can edit the name of the data set on the General page and modify the fields on the Fields page.

You can edit the populated fields, delete them by using the Remove (X) icon, or add new ones by using the Add (+) icon above the Fields list. Any fields you add in this list show up in the Report Explorer tree view and you can drag and drop them onto the design surface. The field name must be unique within the dataset.

When working with Fields, the meaning of the value varies depending on the data source type. In most cases this is simply

the name of the field. The following table describes the meaning of the field value and gives some examples of how to use the value.

Data Provider	Description	Example
SQL, OleDb	The field value is the name of a field returned by the query.	Query: OrderQuantity FirstName
Dataset	The field value can be the name of a field in the DataTable specified by the query. You can also use DataRelations in a DataSet, specify the name of the relation followed by a period and then the name of a field in the related DataTable.	Query: Quantity OrdersToOrderDetails.CustomerID
XML	The field value is an XPath expression that returns a value when evaluated with the query.	Query: Statistics/Game/TeamName
JSON	The field value is a JSONPath expression that returns a value when evaluated with the query.	Query: $.Statistics.Game[*].TeamName
Object	The field value can be the name of a property of the object contained in the collection returned by the data provider. You may also use properties available for the object returned from a property.	Query: Quantity Order.Customer.FirstName
CSV	The field value is the name of a field returned by each column specified in the connection string.	Connection string: Path=C:\\Data\\FixedWidth.csv;Locale=en-US;TextQualifier=";ColumnsSeparator=,;RowsSeparator=\r\n;HasHeaders=True

Parameters

The **Parameters** page of the Dataset dialog is where you can pass a Report Parameter into the parameter you enter in the **Query** page. Enter a Name that matches the name of the Report Parameter and a Value for each parameter in this page.

- You can edit the parameters by selecting a parameter in the list and editing its **Name** and **Value**.
- You can delete the parameters by using the Remove (X) icon above the Parameters list.
- You can add new parameters by using the Add (+) icon above the Parameters list. The parameter name must be unique within the dataset.

The Value of a parameter can be a static value or an expression referring to an object within the report. The Value cannot refer to a report control or field.

Filters

The **Filters** page of the Dataset dialog allows you to filter data after it is returned from the data source. This is useful when you have a data source (such as XML) that does not support query parameters.

A filter is composed of three fields:

Expression: Type or use the expression editor to provide the expression on which to filter data.

Operator: Select from the following operators to decide how to compare the expression to the left with the value to the right:

- **Equal** Only choose data for which the value on the left is equal to the value on the right.
- **Like** Only choose data for which the value on the left is similar to the value on the right. For more information on using the Like operator, see the MSDN Web site.
- **NotEqual** Only choose data for which the value on the left is not equal to the value on the right.
- **GreaterThan** Only choose data for which the value on the left is greater than the value on the right.
- **GreaterThanOrEqual** Only choose data for which the value on the left is greater than or equal to the value on the right.
- **LessThan** Only choose data for which the value on the left is less than the value on the right.
- **LessThanOrEqual** Only choose data for which the value on the left is less than or equal to the value on the right.
- **TopN** Only choose items from the value on the left which are the top number specified in the value on the right.
- **BottomN** Only choose items from the value on the left which are the bottom number specified in the value on the right.
- **TopPercent** Only choose items from the value on the left which are the top percent specified in the value on the right.
- **BottomPercent** Only choose items from the value on the left which are the bottom percent specified in the value on the right.
- **In** Only choose items from the value on the left which are in the array of values specified on the right. Selecting this operator enables the Values list at the bottom.
- **Between** Only choose items from the value on the left which fall between the pair of values you specify on the right. Selecting this operator enables two Value boxes instead of one.

Value: Enter a value to compare with the expression on the left based on the selected operator. For multiple values used with the **Between** operator, the lower two value boxes are enabled.

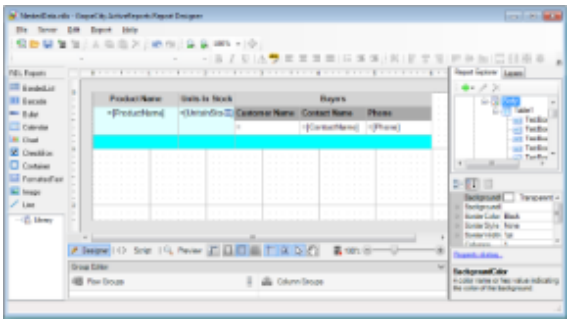
Values: When you choose the **In** operator, you can enter as many values as you need in this list.

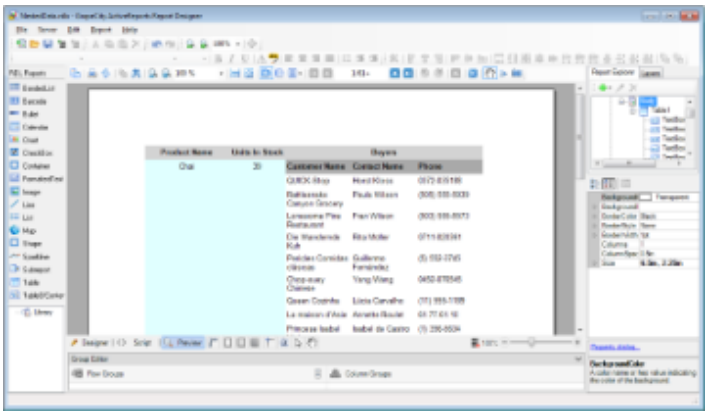
Nested Data Regions Bound to Different Data

In Page and RDL reports, you can use nested data regions that are bound to different datasets. To display data, you can either use a **filter** for a nested data region or a **parameter** that is set in the new **GrapeCity.ActiveReports.PageReportModel.DataRegion.DataSetParameters** property.

Binding data regions to different data is available for all data regions that you can use in Page and RDL Reports, that is Tablix, List, Chart, BandedList, Table, and Sparkline.

The report below shows the customer's contact name and phone information for products that are still in stock. The report layout consists of three (3) nested Table data regions, each data region bound to a different dataset - **Products** (Table1), **Invoices** (Table2), and **Customers** (Table3).





Using a filter

To display data in nested data regions that are bound to different data, you can set a filter in the **Table - Filters** dialog. This filter will contain the value from a nested data region in the left side and the value from a parent data region in the right side of it.

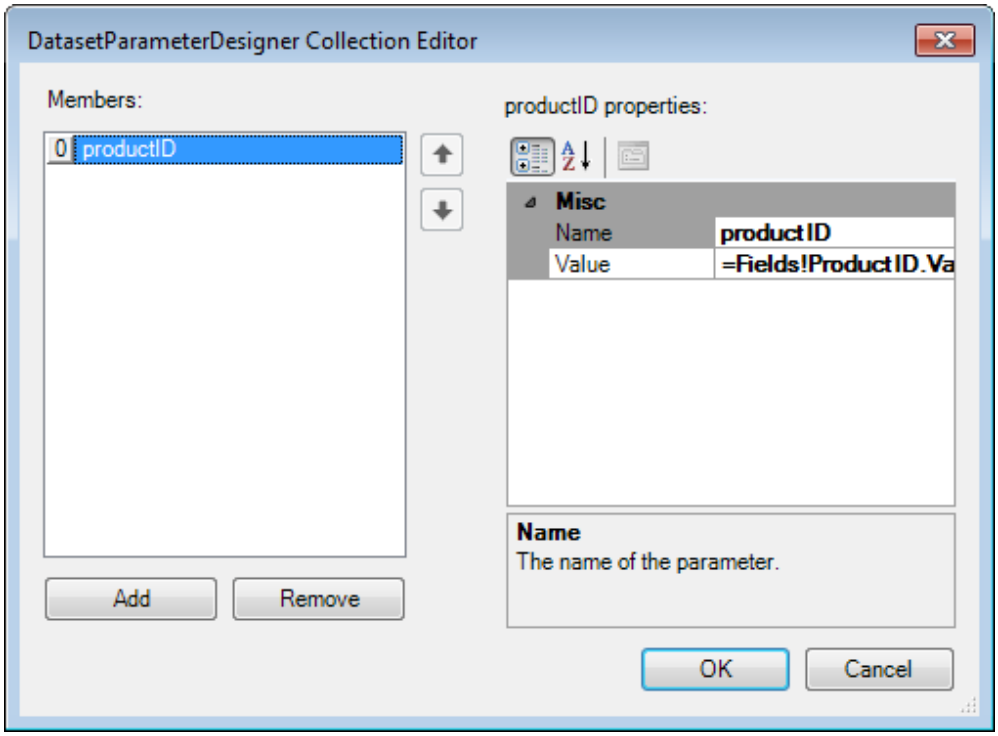
In the sample report above, two (2) filters are created. For the Table2 data region bound to the **Invoices** dataset, we create the filter with the expression [ProductID]=[ProductID]. For the Table3 data region bound to the **Customers** dataset, we create the filter with the expression [CustomerID]=[CustomerID].

As a result, the report shows the Product Name and Units In Stock information from the **Products** dataset. For each Product Name, the report shows the Customer Name information from the **Invoices** dataset and the Contact Name and Phone information from the **Customers** dataset.

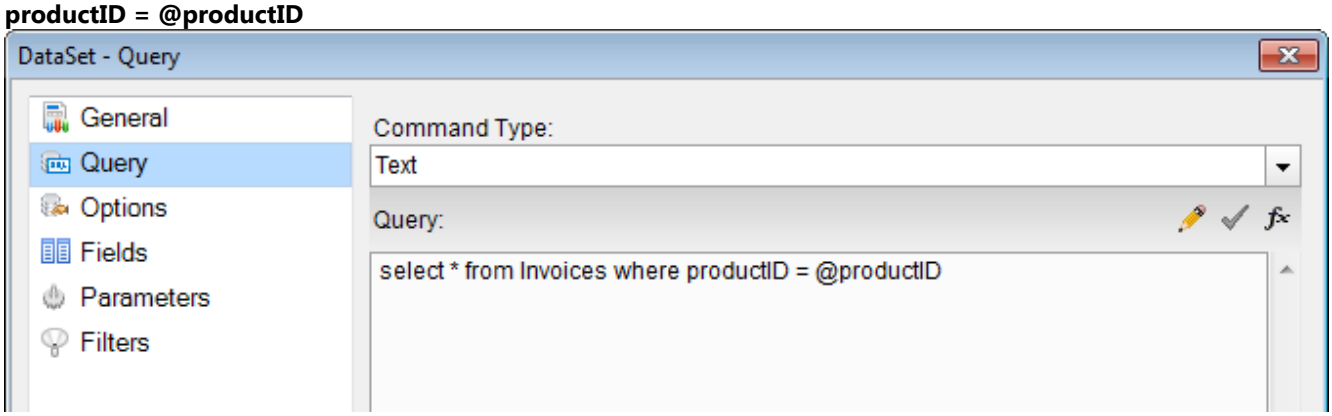
Using a parameter

Setting a parameter in the **DataSetParameters** property also allows displaying data in nested data regions that are bound to different datasets. The basic steps are as follows.

1. In the **DataSetParameters** property of a data region, add a new parameter, for example, **productID**.



2. In the data region's dataset, add a new parameter - **productID**.
3. In the data region's dataset, add a substring with the parameter to the existing dataset query, for example: **WHERE productID = @productID**



For the sample report layout above, two parameters are created. For the Table2 data region, bound to the **Invoices** dataset, we create the parameter **productID** and modify the dataset query as *select * from Invoices where productID = @productID*. For the Table3 data region, bound to the **Customers** dataset, we create the parameter **customerID** and modify the dataset query as *select * from Invoices where customerID = @customerID*.

As a result, the report shows the Product Name and Units In Stock information from the **Products** dataset. For each Product Name, the report shows the Customer Name information from the **Invoices** dataset and the Contact Name and Phone information from the **Customers** dataset.

Expressions

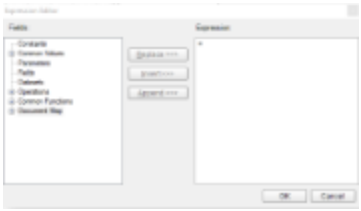
In ActiveReports, you can use an expression to set the value of a control in the report, or set conditions under which certain styles apply. You can set Microsoft Visual Basic® .NET in expressions through,

- Properties in the properties window
- Expression Editor dialog

All expressions begin with an equal sign (=). Even the expression for a field value for a TextBox is set as follows:
`=Fields!LastName.Value`

Expression Editor Dialog

You can build expressions quickly using the Expression Editor dialog. This dialog allows you to choose from a number of fields available to the report as well as to a particular property. You can access the Expression Editor by selecting nearly any property of a control and choosing **<Expression...>** from the drop-down list.



There are the following types of fields available in the Expression Editor:

- **Constants**
  Constants available for properties which have enumerated values such as TextDecoration or BorderStyle.
- **Common Values**
  Run time values available to every property in every report. There are two variables in this list which come from the User collection: User ID and User Language. See Common Values for further information.
- **Parameters**
  Parameters fields available in reports which contain report parameters. If available, you can choose a parameter from this field to retrieve the current value of the parameter.
- **Fields (*DataSet name*)**
  All fields from a dataset which is linked to the report control.
- **Datasets**
  All fields in each dataset associated with the report. However, the report retrieves only the sum or the first value of any field that is not within the current dataset scope.
- **Operations**
  Arithmetic, comparison, concatenation, logical/bitwise, bit shift operators for creating custom expressions.
- **Common Functions**
  Predefined Visual Basic .NET functions for which ActiveReports provides intrinsic support. See Common Functions for more information.
- **Document Map**
  The DocumentMap.Path expression defines labels for the report's TableOfContents members. The example of such expression is =DocumentMap.Path & " General Information". If this expression is defined in the **Label** property of the report's control associated with the report's TableOfContents, **General Information** will be displayed as the label of the corresponding report's TableOfContents member.

To create an Expression in the Expression Editor

The Expression Editor dialog is composed of two panes, Fields and Expression.

- From the Fields pane, select a field you want to use in your expression.
- Click the Replace, Insert or Append button to add the field to the Expression pane. The expression pane shows the fields in a valid expression format.

- Click OK to close the dialog.

The expression appears as the property value in the properties grid.

> **Tip:** While building an expression, you can directly add the entire expression or part of it in the Expression pane of the Expression Editor. Then use the Insert or Append buttons to create a complete expression.

Using Expressions in Reports

In the raw form, your data may not be ideally suited for display in a report. You can customize it and bring it into shape using expressions. Following are some examples of how expressions are set in different scenarios.

Concatenating Fields and Strings

You can concatenate fields with strings and with other fields. For e.g., use the following expression to get a result like **Customer Name: Bossert, Lewis**.

```
="Customer Name: " & Fields!LastName.Value & "," & Fields!FirstName.Value
```

Conditional Formatting

You can use expressions in properties like Color, Font, Border etc. on specific field values based on a condition, to highlight a part of data. The formula for conditional formatting is:
=iif(Fields!*YourFieldName*.Value *operator* "*Value to compare*", "*If condition is met, use this value.*", "*If not, use this one.*"

For e.g., if you enter the following expression in the **Font > FontWeight** property of a textbox that displays names of people, you get the name "Denise" in bold.

```
=iif(Fields!FirstName.Value = "Denise", "Bold", "Normal")
```

Functions

You can use a number of aggregate and other functions in your expressions. ActiveReports includes a range of functions, including running value, population standard variance, standard deviation, count, minimum and maximum. For e.g., use the following expression to get a count of employees.

```
=Count(Fields!EmployeeID.Value, Nothing)
```

Displaying Expressions at Design Time

As you design the report, the full text of an expression can get very long. ActiveReports makes expressions easier to read by shortening them.

When an expression is in the form:
=Fields!<FieldName>.Value

On the design surface, you see this text inside that TextBox:
=[<FieldName>]

Double-click the TextBox to view the full expression in edit mode.

For aggregates too, when the Expression value is:
=<Aggregate>(Fields!<FieldName>.Value)

On the design surface, you see this text inside the TextBox:
=<Aggregate>([<FieldName>])

This shortened expression value is only a visual change to allow you to see the field name easily. It shows up in both the TextBox on the design surface as well as any dropdown boxes inside the dialogs.

> **Note:** You can type the format as listed above for either field name values or aggregates on field names. This evaluates the full expression when the report is viewed.

Besides the shorthand for field names, you can also type shorthand like [@Param] for parameters and [&Value] for Globals such as [&PageNumber] on the design surface. Please note that you cannot use shorthand in the Expression Editor.

Common Values

Common Values are run time values available to every property in every report. You can directly drag and drop these common values from the Report Explorer onto the design surface or add and modify the values from the Expression Editor. Following is a list of the values that you can see under the Common Values node in the Report Explorer and the Expression Editor.

Value	Description	Expression
Page N of M	Gets both the current page and the total number of pages in the report.	="Page " & Globals!PageNumber & " of" & Globals!TotalPages
Page N of M (Section)	Gets both the current page and the total number of pages in the report section.	="Page " & Globals!PageNumberInSection & " of " & Globals!TotalPagesInSection
Page N of M (Cumulative)	Gets both the current page and the total number of cumulative pages in a report.	="Page " & Globals!CumulativePageNumber & " of " & Globals!CumulativeTotalPages
Current Date and Time	Gets the date and time when the report began to run.	=Globals!ExecutionTime
User ID	Gets the machine name/user name of the current user.	=User!UserID
Page Number	Gets the current page number in the report.	=Globals!PageNumber
Page Number (Section)	Gets the current page number in the report section.	=Globals!PageNumberInSection
Total Pages	Gets the total number of pages in the report.	=Globals!TotalPages
Total Pages (Section)	Gets the total number of pages in the report section.	=Globals!TotalPagesInSection

Cumulative Page Number	Gets the current cumulative page number.	=Globals!CumulativePageNumber
Cumulative Total Pages	Gets the total number of cumulative pages in the report.	=Globals!CumulativeTotalPages
Report Folder	Gets the name of the folder containing the report.	=Globals!ReportFolder
Report Name	Gets the name of the report.	=Globals!ReportName
User Language	Gets the language settings of the current user.	=User!Language

Note: Page N of M (Section), Page Number (Section) or Total Pages (Section) is applied to page numbering when you set grouping in a report. Each section represents a group, not to be confused with sections in a section report.

Note: Page N of M (Cumulative), Page Number (Cumulative) or Total Pages (Cumulative) is applied to page numbering when you use collation in a report.

Common Functions

You can use a function in an expression to perform actions on data in data regions, groups and datasets. You can access these functions in the Expression Editor dialog. In any property that accepts expressions, you can drop down the property and select **<Expression...>** to open the dialog.

Within the Expression Editor dialog, there is a tree view of Fields. Expand the **Common Functions** node to view the available functions. The following tables contain details about each of the functions included in ActiveReports for use in property expressions.

Date & Time

These are all methods from the DateAndTime class in Visual Basic. Please see the msdn DateAndTime Class topic for information on overloads for each method.

These are all the available aggregate functions:

Function	Description	Syntax and Example
DateAdd	Returns a date and time value that is the result of adding the interval to the date and time field of the specified unit.	`DateAdd(<DateInterval>,<Number>,<DateTime>)` `=DateAdd("d", 5, Fields!SaleDate.Value); =DateAdd(DateInterval.Day, 5, Fields!SaleDate.Value)`
DateDiff	Returns the difference between the start date and time and end date and time of the specified unit.	`DateDiff(<DateInterval>,<DateTime1>,<DateTime2>[,<DayOfWeek>[,WeekOfYear]]))` `=DateDiff("yyyy"), Fields!SaleDate.Value,"1/1/2015");=DateDiff(DateInterval.Year,Fields!SaleDate.Value,"1/1/2015")`
DatePart	Returns the Integer value that represents the specified part of the given date.	`DatePart(<DateInterval>,<DateTime1>[,<FirstDayOfWeek>[,FirstWeekOfYear]]))` `=DatePart("m", Fields!SaleDate.Value)`
DateSerial	Returns a Date	`DateSerial(<Year Number>,<Month Number>,<Day Number>)`

	value that represents a specified year, month, and a day, with the time information set to midnight (00:00:00).	`=DateSerial(DatePart("yyyy", Fields!SaleDate.Value)-10, DatePart("m", Fields!SaleDate.Value)+5, DatePart("d", Fields!SaleDate.Value)-1)`
DateString	Returns the String value that represents the current date in your system.	`DateString()` `=DateString()`
DateValue	Returns a Date value that contains the information on date represented by a string, with the time set to midnight (00:00:00).	`DateValue(<StringDate>)` `=DateValue("December 12, 2015")`
Now	Returns the current date and time in your system.	`Now()` `=Now()`
Today	Returns a Date value that contains the current date in your system.	`Today()` `=Today()`
Day	Returns an Integer value from 1 through 31 that represents the day of the month.	`Day(<DateTime>)` `=Day(Fields!SaleDate.Value)`
Hour	Returns an Integer value from 0 through 23 that represents the hour of the day.	`Hour(<DateTime>)` `=Hour(Fields!SaleDate.Value)`
Minute	Returns an Integer value from 0 through 59 that represents the minute of the hour.	`Minute(<DateTime>)` `=Minute(Fields!SaleDate.Value)`
Month	Returns an Integer value from 0 through 12 that represents the month of the year.	`Month(<DateTime>)` `=Month(Fields!SaleDate.Value)`
MonthName	Returns the name of the month specified in the date as a String.	`MonthName(<Month Number>[,<Abbreviate>])` `=MonthName(Fields!SaleDate.Value)`
Second	Returns an	`Second(<DateTime>)`

	Integer value from 0 through 59 that represents the second of the minute.	`=Second(Fields!SaleDate.Value)`
TimeSerial	Returns a Date value that represents a specified hour, minute, and second, with the date information set relative to January 1 of the year 0001.	`TimeSerial(<Hour Number>, <Minute Number>, <Second Number>)` `=TimeSerial(DatePart("h", Fields!SaleDate.Value), DatePart("n", Fields!SaleDate.Value), DatePart("s", Fields!SalesDate.Value))`
TimeValue	Returns a Date value that contains the information on time represented by a string, with the date set to January 1 of the year 0001.	`TimeValue(<StringTime>)` `=TimeValue("15:25:45"); TimeValue(Fields!SaleDate.Value)`
TimeOfDay	Returns a Date value containing the current time of day in your system.	`TimeOfDay()` `=TimeOfDay()`
Timer	Returns a Double value that represents the number of seconds elapsed since midnight.	`Timer()` `=Timer()`
TimeString	Returns the String value that represents the current time of day in your system.	`TimeString()` `=TimeString()`
Weekday	Returns an Integer value that contains a number representing the day of the week.	`Weekday(<DateTime[,<DayOfWeek>])` `=Weekday(Fields!SaleDate.Value,0)`
WeekdayName	Returns a String value that contains the name of the specified weekday.	`WeekdayName(<WeekDay>[,<Abbreviate[, <FirstDayOfWeek>]])` `=WeekdayName(3, True, 0); =WeekDayName("w", Fields!SaleDate.Value), True, 0)`
Year	Returns an Integer value from 1 through 9999 representing the year.	`Year(<DateTime>)` `=Year(Fields!SaleDate.Value)`
Quarter	Returns an	`Quarter(<DateTime>)`

	Integer value from 1 through 4 representing the quarter number.	`=Quarter(Fields!SaleDate.Value)`
QuarterName	Returns a string value representing the quarter name.	`QuarterName(<DateTime>)` `=QuarterName(Fields!SaleDate.Value)`

Math

These are all methods and fields from the System.Math class. Please see the msdn Math Class topic for information on overloads for each method.

Function	Description	Syntax and Example
Abs	Returns the absolute or positive value of a single-precision floating-point number.	`Abs(<Number>)` `=Abs(-5.5);=Abs(Fields!YearlyIncome.Value-80000)`
Acos	Returns the angle whose cosine is the specified number.	`Acos(<Number>)` `=Acos(.5); =Acos(Fields!Angle.Value)`
Asin	Returns the angle whose sine is the specified number	`Asin(<Number>)` `=Asin(.5); =Asin(Fields!Angle.Value)`
Atan	Returns the angle whose tangent is the specified number.	`Atan(<Number>)` `=Atan(.5); =Atan(Fields!Angle.Value)`

Atan2	Returns the angle whose tangent is the quotient of two specifed numbers.	`Atan2(<Number1>,<Number2>)`
		`=Atan2(3,7);`
		`=Atan2(Fields!CoordinateY.Value,Fields!CoordinateX.Value)`

BigMul	Returns the multiplication of two 32-bit numbers.	`BigMul(<Number1>,<Number2>)`
		`=BigMul(4294967295,-2147483647);`
		`=BigMul(Fields!Int32Value.Value, Fields!Int32Value.Value)`

Ceiling	Returns the smallest integer greater than or equal to the specified double-precision floating-point number.	`Ceiling(<Number>)` `=Ceiling(98.4331); =Ceiling(Fields!AnnualSales.Value /6)`
Cos	Returns the smallest integer greater than or equal to the specifed double-precision floating-point number.	`Cos(<Number>)` `=Cos(60)`
Cosh	Returns the hyperbolic cosine of the specified angle.	`Cosh(<Number>)` `=Cosh(60)`
E	Returns the value of E, which is 2.71828182845905.	`E` `=E*2`
Exp	Returns e raised to the specified ^, where is Euler s number. It is the inverse of the Log function.	`Exp(<Number>)` `=Exp(3); =Exp(Fields!IntegerCounter.Value)`
Fix	Returns the integer portion of a number.	`Fix(<Number>)` `=Fix(-7.15); =Fix(Fields!AnnualSales.Value /-5)`
Floor	Returns the longest integer less than or equal to the specified double-precision floating-point number.	`Floor(<Number>)` `=Floor(4.67); =Floor(Fields!AnnualSales.Value/ 12)`
IEEERemainder	Returns the remainder after division of one number by another according to IEEE satndards.	`IEEERemainder(<Number1>,<Number2>)` `=IEEERemainder(9,8)`
Log	Returns the logarithm of the specified number.	`Log(<Number>)` `=Log(20.5); =Log(Fields!NunberValue.Value)`
Log10	Returns the logarithm of the specified number to the base	`Log10(<Number>)`

	10.	`=Log10(20.5); =Log10(Fields!NumberValue.Value)`
Max	Returns the maximum non-null value from the specified expression.	`Max(<Value>)` `=Max(Fields!OrderTotal.Value)`
Min	Returns the minimum non-null value from the specified expression.	`Min(<Value>)` `=Min(Fields!OrderTotal.Value)`
PI	Returns the value of PI, which is 3.14159265358979.	`PI` `=2 * PI * Fields!Radius.Value`
Pow	Returns one number raised to the ^ of another number.	`Pow(<Number1,<Number2>)` `=Pow(Fields!Quantity.Value, 2)`
Round	Returns the round-off of a decimal number to the nearest integer or to the nearest decimal number up to the specified digits.	`Round(<Number>)` `=Round(12.456); =Round(Fields!AnnualSales.Value / 12.3)`
Sign	Returns a value indicating the sign of an 8-bit signed integer.	`Sign(<Number>)` `=Sign(Fields!AnnualSales.Value-60000)`
Sin	Returns the sine of the specified number.	`Sin(<Number>)` `=Sin(60)`
Sinh	Returns the hyperbolic sine of the specified angle.	`Sinh(<Number>)` `=Sinh(60)`
Sqrt	Returns the square root of the specified number.	`Sqrt(<Number>)` `=Sqrt(121)`
Tan	Returns the tangent of the specified number.	`Tan(<Number>)` `=Tan(60)`
Tanh	Returns the hyperbolic tangent of the specified angle.	`Tanh(<Number>)` `=Tanh(60)`

Inspection

These are all methods from the DateAndTime class in Visual Basic. Please see the msdn DateAndTime Class topic for information on overloads for each method.

Function	Description	Syntax and Example
IsArray	Returns True if the expression can be evaluated as an array.	`IsArray(<Expression>)` `=IsArray(Parameters!Initials.Value)`
IsDate	Returns True if the expression represents a valid Date value.	`IsDate(<Expression>)` `=IsDate(Fields!BirthDate.Value); =IsDate("31/12/2010")`
IsDBNull	Returns True if the expression evaluates to a null.	`IsDBNull(<Expression>)` `=IsDBNull(Fields!MonthlySales.Value)`
IsError	Returns True if the expression evaluates to an error.	`IsError(<Expression>)` `=IsError(Fields!AnnualSales.Value = 80000)`
Isnothing	Returns True if the expression evaluates to nothing.	`IsNothing(<Expression>)` `=IsNothing(Fields!MiddleInitial.Value)`
IsNumeric	Returns True if the expression can be evaluated as a number.	`IsNumeric(<Expression>)` `=IsNumeric(Fields!AnnualSales.Value)`

ProgramFlow

These are all methods from the Interaction class in Visual Basic. Please see the msdn Interaction Class topic for more information.

Function	Description	Syntax and Example
Choose	Returns a value from a list of arguments.	`Choose(<Index>,<Value>[, <Value2>,...[, <Value N>]])` `=Choose(3, "10", "15", "20", "25")`
IIF	Returns the value if the expression evaluates to True, and the second value if the expression evaluates to False.	`IIF(<Condition>, <TruePart>, <FalsePart>)` `=IIF(Fields!AnnualSales.Value >= 80000, "Above Average", "Below Average")`
Partition	Returns a string (in the form x : y) that represents the calculated range based on the specified interval containing the specified number.	`Partition(<Value>, <Start>, <End>, <Interval>)` `=Partition(1999, 1980, 2000, 10)`
Switch	Returns the value of the first expression that evaluates to True among a list of expressions.	`Switch(<Condition1>, <Value1>[, <Condition2>, <Value2>,...[, <ConditionN>, <ValueN>]])` `=Switch(Fields!FirstName.Value = "Abraham", "Adria", Fields!FirstName.Value = "Charelotte", "Cherrie")`

Aggregate

You can use aggregate functions within report control value expressions to accrue data. ActiveReports supports aggregate functions from RDL 2005, plus some proprietary extended set of functions. For all of the functions, you can add an optional <Scope> parameter.

These are all the available aggregate functions:

Function	Description	Syntax and Example
AggregateIf	Decides whether to calculate a custom aggregate from the data provider of the values returned by the expression based on a Boolean expression.	`AggregateIf(<Condition>, <AggregateFunction>, <AggregateArguments>)` `=AggregateIf(Fields!Discontinued.Value=True, Sum, Fields!InStock.Value)`
Avg	Calculates the average of the non-null values returned by the expression.	`Avg(<Values>)` `=Avg(Fields!Cost.Value, Nothing)`
Count	Calculates the number of non-null values returned by the expression.	`Count(<Values>)` `=Count(Fields!EmployeeID.Value, Nothing)`
CountDistinct	Calculates the number of non-repeated values returned by the expression.	`CountDistinct(<Values>)` `=CountDistinct(Fields!ManagerID.Value, "Department")`
CountRows	Calculates the number of rows in the scope returned by the expression.	`CountRows()` `=CountRows("Department")`
CumulativeTotal	Calculates the sum of page-level aggregates returned by the expression for current and previous pages.	`CumulativeTotal(<Expression>, <Aggregate>)` `=CumulativeTotal(Fields!OrderID.Value, Count)`
DistinctSum	Calculates the sum of the values returned by an expression using only the rows when the value of another expression is not repeated.	`DistinctSum(<Values>, <Value>)` `=DistinctSum(Fields!OrderID.Value, Fields!OrderFreight.Value, "Order")`
First	Shows the first value returned by the expression.	`First(<Values>)` `=First(Fields!ProductNumber.Value, "Category")`
Last	Shows the last value returned by the expression.	`Last(<Values>)` `=Last(Fields!ProductNumber.Value, "Category")`
Max	Shows the largest non-null value returned by the expression.	`Max(<Values>)` `=Max(Fields!OrderTotal.Value, "Year")`
Median	Shows the value that is the mid-point of the values returned by the expression. Half of the values returned will be above this value and half will be below it.	`Median(<Values>)` `=Median(Fields!OrderTotal.Value)`

Min	Shows the smallest non-null value returned by the expression	`Min(<Values>)` `=Min(Fields!OrderTotal.Value)`
Mode	Shows the value that appears most frequently in the values returned by the expression.	`Mode(<Values>)` `=Mode(Fields!OrderTotal.Value)`
RunningValue	Shows a running aggregate of values returned by the expression (Takes one of the other aggregate functions as a parameter),	`RunningValue(<Values>, <AggregateFunction>)` `=RunningValue(Fields!Cost.Value, Sum, Nothing)`
StDev	Calculates the dispersion (standard deviation) of all non-null values returned by the expression.	`StDev(<Values>)` `=StDev(Fields!LineTotal.Value, "Order")`
StDevP	Calculates the population dispersion (population standard deviation) of all non-null values returned by the expression.	`StDevP(<Values>)` `=StDevP(Fields!LineTotal.Value, "Order")`
Sum	Calculates the sum of the values returned by the expression.	`Sum(<Values>)` `=Sum(Fields!LineTotal.Value, "Order")`
Var	Calculates the variance (standard deviation squared) of all non-null values returned by the expression.	`Var(<Values>)` `=Var(Fields!LineTotal.Value, "Order")`
VarP	Calculates the population variance (population standard deviation squared) of all non-null values returned by the expression.	`VarP(<Values>)` `=VarP(Fields!LineTotal.Value, "Order")`

Conversion

These are all methods from the Convert class in the .NET Framework. Please see the msdn Convert Class topic for more information.

Function	Description	Syntax and Example
ToBoolean	Converts the specified value to Boolean.	`ToBoolean(<Value>)` `=ToBoolean(Fields!HouseOwnerFlag.Value)`
ToByte	Converts the specified value to Byte.	`ToByte(<Value>)` `=ToByte(Fields!ProductNumber.Value)`
ToDateTime	Converts the specified value to a Date and Time value.	`ToDateTime(<Value>)` `=ToDateTime(Fields!SaleDate.Value); =ToDateTime("1 January, 2017")`
ToDouble	Converts the specified value to Double.	`ToDouble(<Value>)` `=ToDouble(Fields!AnnualSales.Value); =ToDouble(535.85 * .2691 * 67483)`
ToInt16	Converts the specified value to a 16-bit signed Integer.	`ToInt16(<Value>)` `=ToInt16(Fields!AnnualSales.Value); =ToInt16(535.85)`
ToInt32	Converts the specified value to a 32-bit signed Integer.	`ToInt32(<Value>)` `=ToInt32(Fields!AnnualSales.Value)`
ToInt64	Converts the specified value to a 64-bit signed Integer.	`ToInt64(<Value>)` `=ToInt64(Fields!AnnualSales.Value)`
ToSingle	Converts the specified value to a single-precision floating-point number.	`ToSingle(<Value>)` `=ToSingle(Fields!AnnualSales.Value); =ToSingle(15.857692134)`
ToUInt16	Converts the specified value to a 16-bit unsigned Integer.	`ToUInt16(<Value>)` `=ToUInt16(Fields!AnnualSales.Value)`
ToUInt32	Converts the specified value to a 32-bit unsigned Integer.	`ToUInt32(<Value>)` `=ToUInt32(Fields!AnnualSales.Value)`
ToUInt64	Converts the specified value to a 64-bit unsigned Integer.	`ToUInt64(<Value>)`

		`=ToUInt64(Fields!AnnualSales.Value)`

Miscelleneous

ActiveReports also offers several functions which do not aggregate data, but which you can use with an IIf function to help determine which data to display or how to display it.

The first four are miscellaneous functions from the RDL 2005 specifications. GetFields is a proprietary function to extend RDL specifications.

Function	Description	Syntax and Example
InScope	Determines whether the current value is in the indicated scope.	`InScope(<Scope>)` `=InScope("Order")`
Level	Returns the level of the current value in a recursive hierarchy.	`Level()` `=Level()`
Previous	Returns the previous value within the indicated scope.	`Previous(<Value>)` `=Previous(Fields!OrderID.Value)`
RowNumber	Shows a running count of all the rows in the scope returned by the expression.	`RowNumber()` `=RowNumber()`

GetFields	Returns an IDictionary<string,Field> object that contains the current contents of the Fields collection. Only valid when used within a data region. This function makes it easier to write code that deals with complex conditionals. To write the equivalent function without GetFields() would require passing each of the queried field values into the method which could be prohibitive when dealing with many fields.	`GetFields()`
		`=Code.DisplayAccountID(GetFields())`
		Custom function. Paste in the Code tab.
		`'Within the Code tab, add this function.`
		`Public Function DisplayAccountID( flds as Object) as Object`
		`  If flds("FieldType").Value = "ParentAccount" Then`
		`    Return flds("AccountID").Value`
		`  Else`
		`    Return flds("ParentAccountID").Value`
		`  End If`
		`End Function`

Lookup	Returns the first matching value for the specified name from the dataset with pairs of name and value. For more information, see Report Builder Functions - Lookup Function.	`Lookup(<SourceExpression>, <DestinationExpression>, <ResultExpression>, <LookupDataset>)` `=Lookup(Fields!ProductID.Value, Fields!ProductID.Value, Fields!Quantity.Value, "DataSet2")`
LookupSet	Returns multiple row values from a specified dataset and can be used for the 1-to-many relationship. For more information, see Report Builder Functions - LookupSet Function.	`LookupSet(source_expression, destination_expression, result_expression, dataset)` `=LookupSet(Fields!CategoryID.Value, Fields!CategoryID.Value, Fields!UnitsInStock.Value, "Products")`
MapPoint	Allows displaying simple data directly on the Map as a map Point Layer.	`MapPoint(<Latitude>, <Longitude>)` `=MapPoint(Fields!Latitude.Value, Fields!Longitude.Value)`

Scope

All functions have a Scope parameter which determines the grouping, data region, or dataset to be considered when calculating the aggregate or other function. Within a data region, the Scope parameter's default value is the innermost grouping to which the report control belongs. Alternately, you can specify the name of another grouping, dataset, or data region, or you can specify **`Nothing`**, which sets it to the outermost data region to which the report control belongs.

The Scope parameter must be a data region, grouping, or dataset that directly or indirectly contains the report control using the function in its expression. If the report control is outside of a data region, the Scope parameter refers to a dataset. If there is only one dataset in the report, you can omit the Scope parameter. If there are multiple datasets, you must specify which one to use to avoid ambiguity.

Note: You cannot set the Scope parameter to `Nothing` outside of a data region.

Expressions in Reports

You can use expressions in the control's properties to calculate values. You can also use expressions to concatenate fields, to concatenate strings with fields, to aggregate data, to set formatting based on field values, to show or hide other controls based on field values and even to display a graphical representation of the data. This topic illustrates the how to use expressions to achieve different effects.

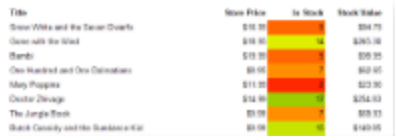
Note:

- This topic uses the **MovieProduct** table from the Reels database. The Reels.mdb file can be downloaded from GitHub: ..\Samples14\Data\Reels.mdb.
- Although this topic uses Page reports, you can also implement this using RDL reports.

When you complete these steps, you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout

Run-Time Layout



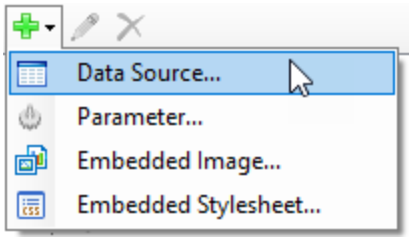
Add an ActiveReport to the Visual Studio project

1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 Page Report** and in the Name field, rename the file as rptExpressions.
4. Click the **Add** button to open a new fixed page report in the designer.

See Quick Start for information on adding different report layouts.

Connect the report to a data source

1. In the Report Explorer, right-click the Data Sources node and select the **Add Data Source** option or select **Data Source** from the Add button.



2. In the Report Data Source Dialog that appears, select the **General** page and in the Name field, enter a name like

ReportData.
3. On this page, create a connection to the Reels database. See Connect to a Data Source for information on connecting to a data source.

Add a dataset

1. In the Report Explorer, right-click the data source node and select the **Add Data Set** option or select **Data Set** from the Add button.
2. In the DataSet Dialog that appears, select the **General** page and enter the name of the dataset as DVDStock. This name appears as a child node to the data source icon in the Report Explorer.
3. On the **Query** page of this dialog, in the **Query** field enter the following SQL query.

SQL Query
`SELECT * FROM DVDStock`

4. Click the **Validate DataSet** icon  at the top right hand corner above the Query box to validate the query.
5. Click **OK** to close the dialog. Your data set and queried fields appear as nodes in the Report Explorer.

Create a layout for the report

1. From the toolbox, drag a Table data region onto the design surface and go to the Properties Window to set the **DataSetName** property to **DVDStock**.
2. Right-click in the column handle at the top of the third column and choose **Insert Column to the Right** to add a fourth column of the same width.
3. Click inside the table to display the row and column handles along the left and top edges of the table and set the column width as follows:

Table Column	Width
TableColumn1	3.5in
TableColumn2	1in
TableColumn3	1in
TableColumn4	1in

4. In the Report Explorer from the DVDStock dataset, drag the following fields into the detail row and set their properties as follows.

Data Field	Column Name
TableColumn1	Title
TableColumn2	StorePrice
TableColumn3	InStock

5. Select detail row cell containing the StorePrice values in the TableColumn2 and in the Properties Window, set the **Format** property to Currency.
6. Select the header row using the row handle to the left and in the Properties Window, set the **FontWeight** property to **Bold**.
7. For an Page report, in the Report Explorer select the Table control and in the Properties window, set the FixedSize property to **6.5in, 7in**.

Add a field expression to a text box to multiply two field values

1. In the detail row of the fourth column, enter the following expression: `= Fields!InStock.Value* Fields!StorePrice.Value`
2. Go to the Properties Window to set the **Format** property of the textbox to Currency formatting.
3. In the header row immediately above this Textbox, enter **Stock Value** for the static label.

Add an Immediate If expression to show or hide a report control

1. Select the cell in which we multiplied two field values (in the detail row of the fourth column) and in the Properties window, expand the **Visibility** property.
2. In the **Hidden** property, enter the following immediate if expression to hide the textbox if there is no stock for the item. `=iif(Fields!InStock.Value=0, True, False)`

Add a Data Visualization expression to display data graphically

The ColorScale3 visualizer function displays a range of colors to indicate minimum, average, and maximum values in the data. See the Data Visualizers topic for further information.

Select the cell in the detail row under the **In Stock** label and in the Properties window, set the **BackgroundColor** property to the following expression:

```
=ColorScale3(Fields!InStock.Value, 0, Avg(Fields!InStock.Value),
Max(Fields!InStock.Value),
"Red", "Yellow", "Green")
```

> **Note:** The parameters of the ColorScale3 function evaluate to Value, Minimum, Average, Maximum, StartColor, MiddleColor and EndColor. Note that aggregate functions (Avg and Max) are used within the ColorScale3 function. See Functions for details on these and other aggregate functions.

View the report

- Click the preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

LookupSet Function in Data Regions

The LookupSet function returns multiple row values from a specified dataset, so you will use this function for the 1-to-many relationship. The fields of the dataset returned by the LookupSet function behave as regular dataset fields that you can use in functions/aggregates within the scope of the data region.

The following data regions can use the LookupSet function in the **Value** property - Tablix, Table, Classic Chart, BandedList, List, and Sparkline.

The basic syntax of the Lookup expression is as follows.

```
LookupSet(<SourceExpression>, <DestinationExpression>, <ResultExpression>,
<LookupSetDataset>)
```

- **SourceExpression**: An expression that is evaluated in the current scope and that specifies the name or key to look up.
- **DestinationField**: An expression that is evaluated for each row in a dataset and that specifies the name or key to match on.
- **ResultExpression**: An expression that is evaluated for the row in the dataset where source_expression =

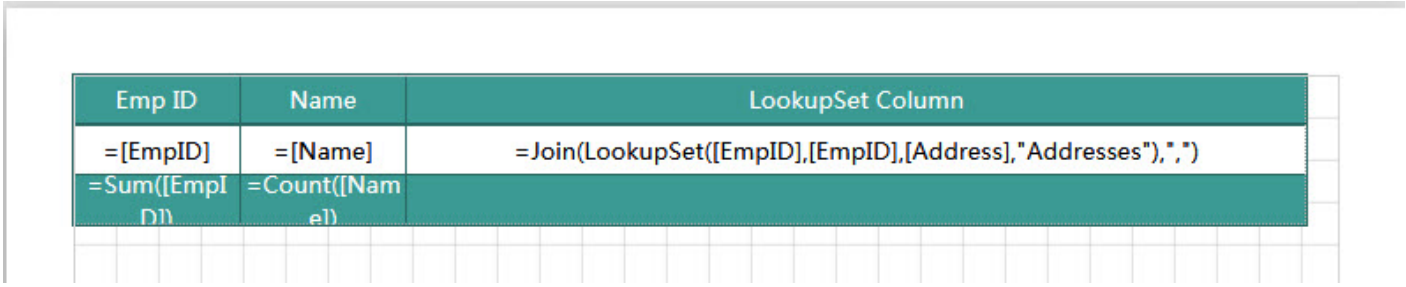
destination_expression, and that specifies the value to retrieve.

- **LookupSetDataset**: A constant that specifies the name of a dataset in the report. For example, "ContactInformation".

The report below shows information on addresses for each employee and displays the addresses as string separated by commas. To display all addresses for each employee in a string separated by commas, we need to use the Join function in the expression with the LookupSet function.

For example:

```
=LookupSet(Fields!CategoryID.Value, Fields!CategoryID.Value, Fields!UnitsInStock.Value,
"Products")
=Join(LookupSet(Fields!CategoryID.Value, Fields!CategoryID.Value,
Fields!UnitsInStock.Value, "Products"), ",")
```



Emp ID	Name	LookupSet Column
=[EmpID]	=[Name]	=Join(LookupSet([EmpID],[EmpID],[Address],"Addresses"),",")
=Sum([EmpID])	=Count([Name])	

Emp ID	Name	LookupSet Column
10001	Davolio	Address01,Address02,Address03
10002	Fuller	Address04
20003	2	

Layers

- **What are Layers?**
- **Why Use Layers?**
- **Other Advantages**

What are Layers?

Layers can be understood as a named group of controls. You can lock or unlock, add or remove, show or hide these groups of controls. When you create a new report, a Default Layer is automatically added to it.

Layers are supported in the following types of reports:

- Page Report
- Rdl Report

Why Use Layers?

You can trace the layout of a pre-printed form accurately using Layers. This feature is useful where you use the scanned copy of the form to trace that you can place on one Layer and you want use it to print.

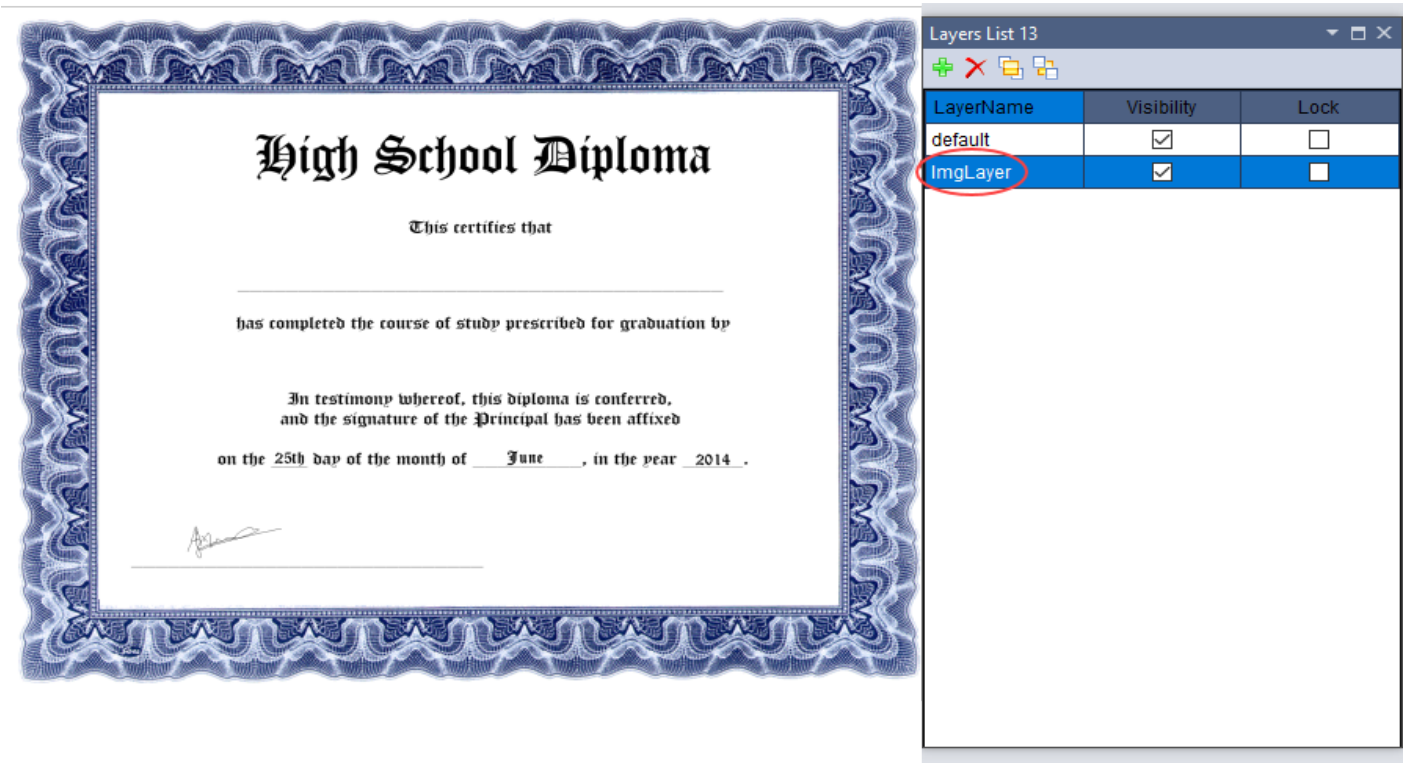
Let us understand this concept with an example of a school diploma certificate. The requirement is to print the name of graduating students on a pre-printed school diploma certificates. We already have a set format for this certificate and a list of names in our data base that need to be printed at the correct location on the certification in the correct style.

Step1: Scan a copy of the school diploma certificate.

The scanned image is placed on a Layer and acts as the base image to identify the location where the name is to be placed.

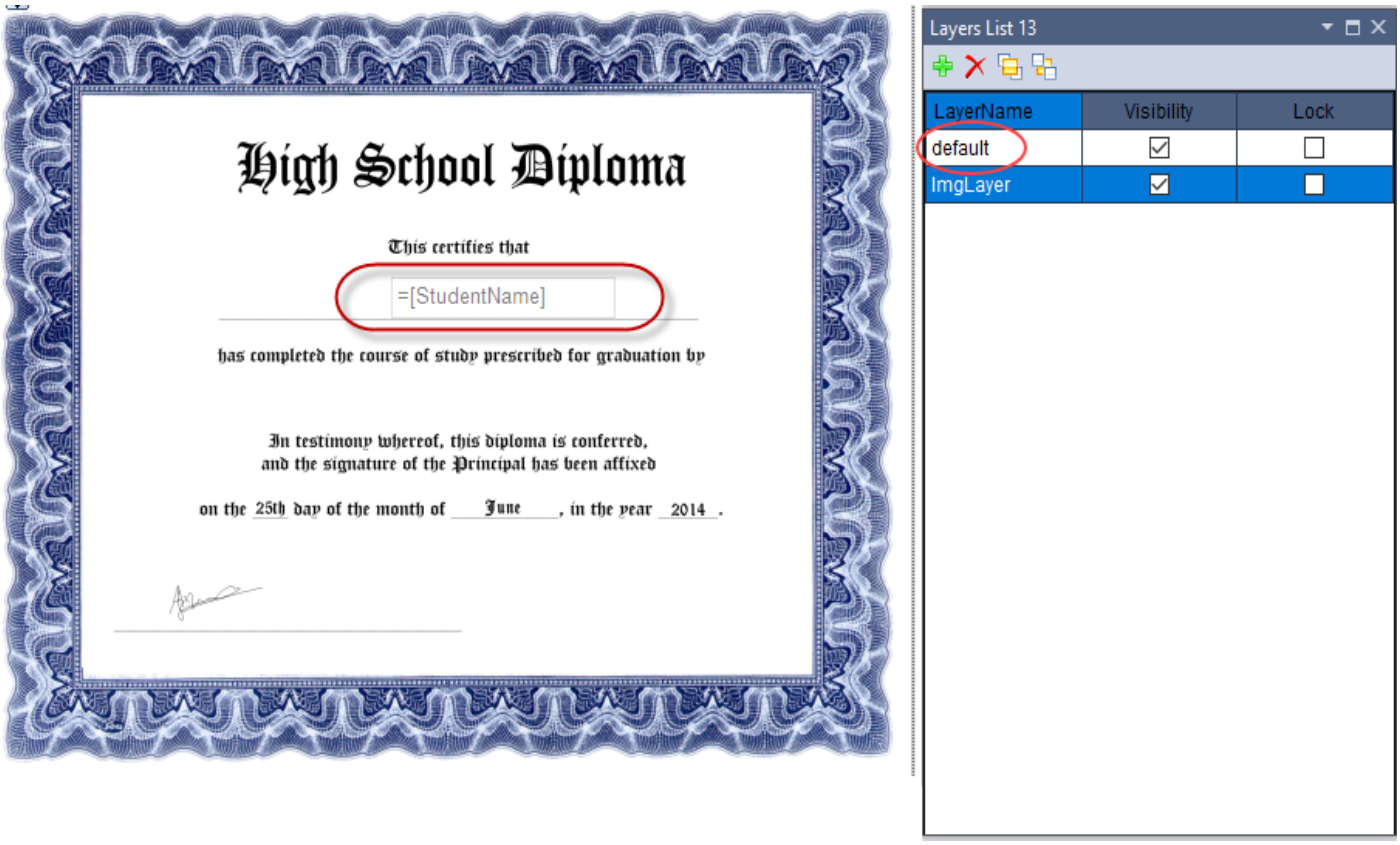
As a best practice, avoid placing images of pre-printed forms on the Default Layer because this Layer cannot be deleted. Instead, create a new Layer to place the scanned image so that you can delete the Layer if you want to remove the scanned image from the background.

The Layer with the image of a pre-printed form is now ready for tracing.



Step 2: Trace a field that contains the names of graduating students.

On the Default Layer, place a TextBox control that is bound to a list of graduating students on the report designer. Placing the **StudentName** field at the accurate location becomes easy with the scanned image Layer displayed in the background.

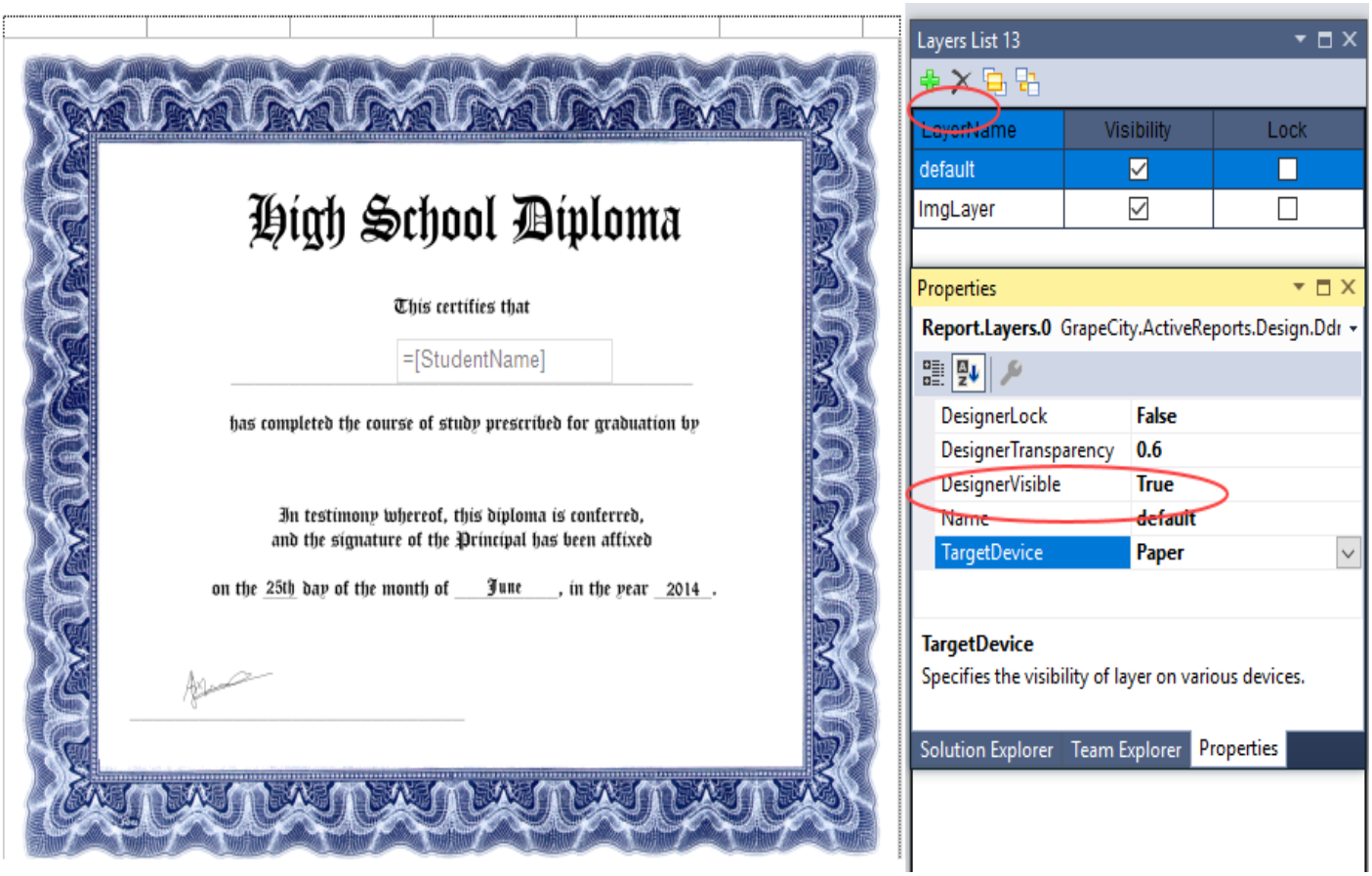


Step 3: Printing the names of graduating students on the school diploma certificate.

Now, that the field has been placed at the correct location and bound to a list of student names, the last step is to print the names on the actual certificates.

Assuming that pre-printed certificates are already placed in the printer, the image Layer that contains the scanned certificate image need not be printed. This can be done using the **TargetDevice** property in Layers.

The TargetDevice property applies to each layer separately and you can choose from **screen**, **paper**, **export**, **all** or **none** options. See View, Export or Print Layers for further details. For this example, set the TargetDevice property of Default Layer to **Paper** for printing the name field on pre-printed certificates.



In this scenario, Layers are used to trace the layout of a field on a pre-printed certificate.

However, Layers are useful in some other scenarios as well.

Other Advantages

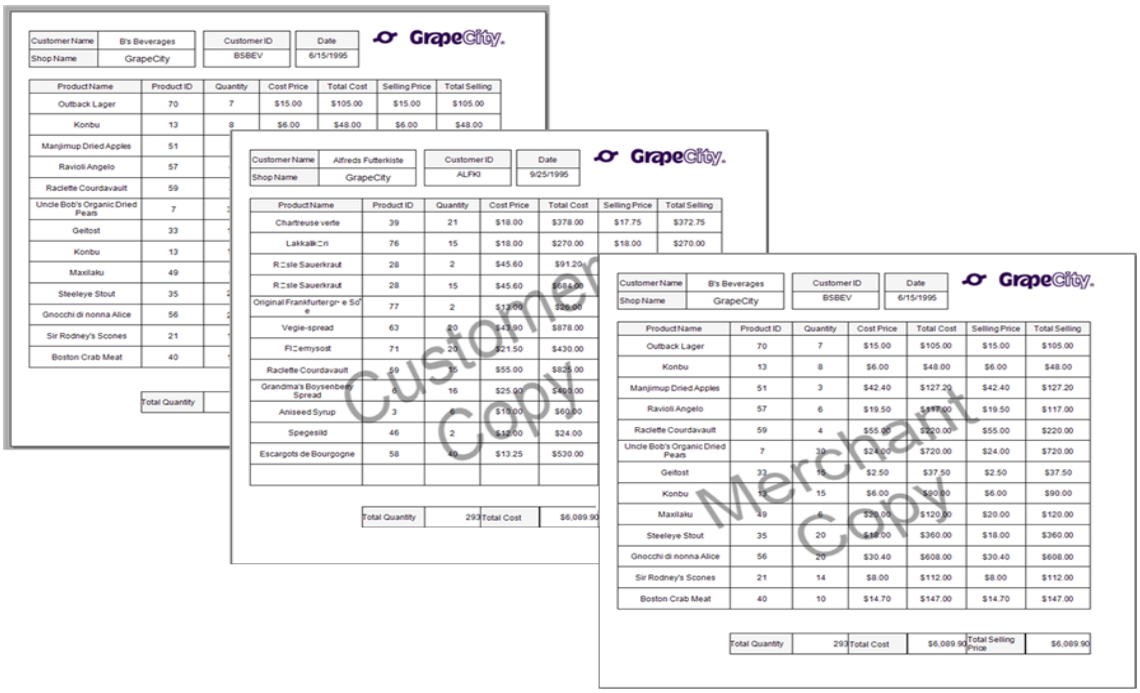
Creating Template Reports

Leverage the advantage of Layers in scenarios where you do not want to make changes to an existing report but want to perform minor modifications to the layout.

With Layers, it is possible to make modifications to the same report without changing the original report layout. Let's take an example of a sales receipt to see how Layers can help in this scenario.

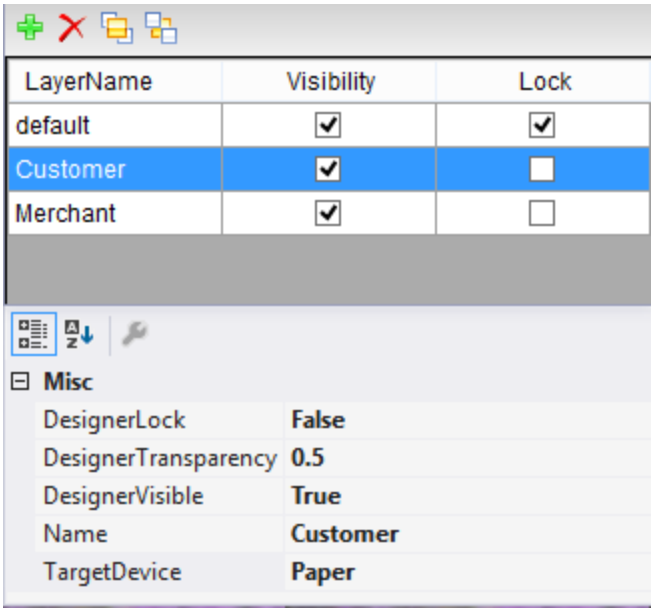
Illustrative Example

The requirement is that a hard copy of the report is printed with a **Customer Copy** watermark and the soft copy of the same report is exported in a PDF format with a **Merchant Copy** watermark.



Lock the Default Layer to use the original sales receipt report as a template. This step is necessary to make sure that the existing layout of the template report is not modified while making changes or adding controls to the layout. Please see Working with Layers for details on how to lock a Layer.

Add two Layers on the existing report template, one for the Customer Copy watermark image and the other for the Merchant Copy watermark image. Set the TargetDevice property of Customer Layer to Paper for printing a hard copy of the sales receipt and Merchant Layer to Export for exporting it to a PDF format. Please see View, Export or Print Layers for details on how to export Layers.



In this scenario an existing report is used as a template to output two different versions of the same report without having to create and save two copies separately.

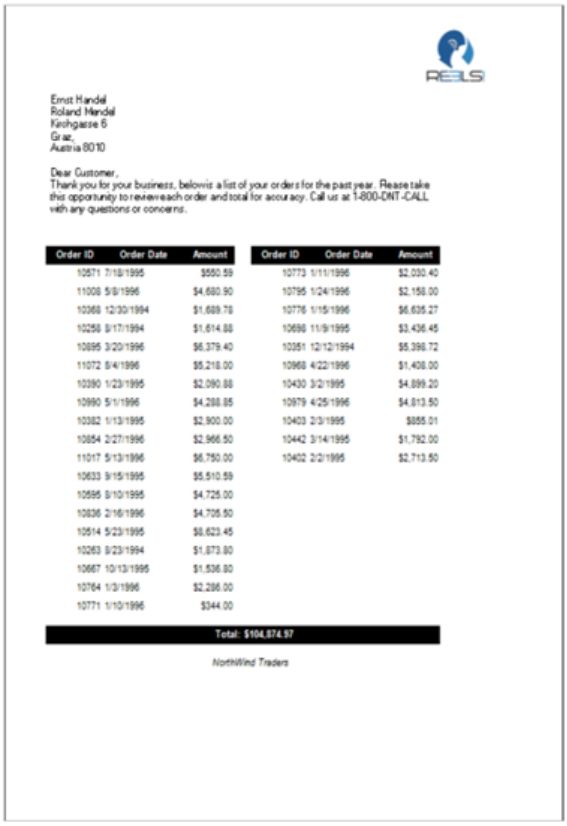
Replicating a Layout

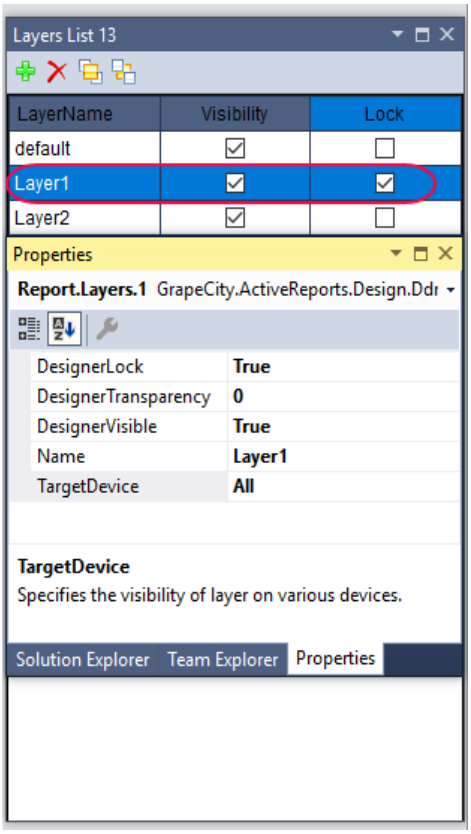
Layers can be used to replicate the layout of a pre-printed form. This is particularly useful when you want to replicate a layout of a pre-printed form which is either uneditable or unavailable in soft copy.

Let us take an example of an order summary letter sent to the customers to see how Layers can help replicate the Layout of the scanned image easily.

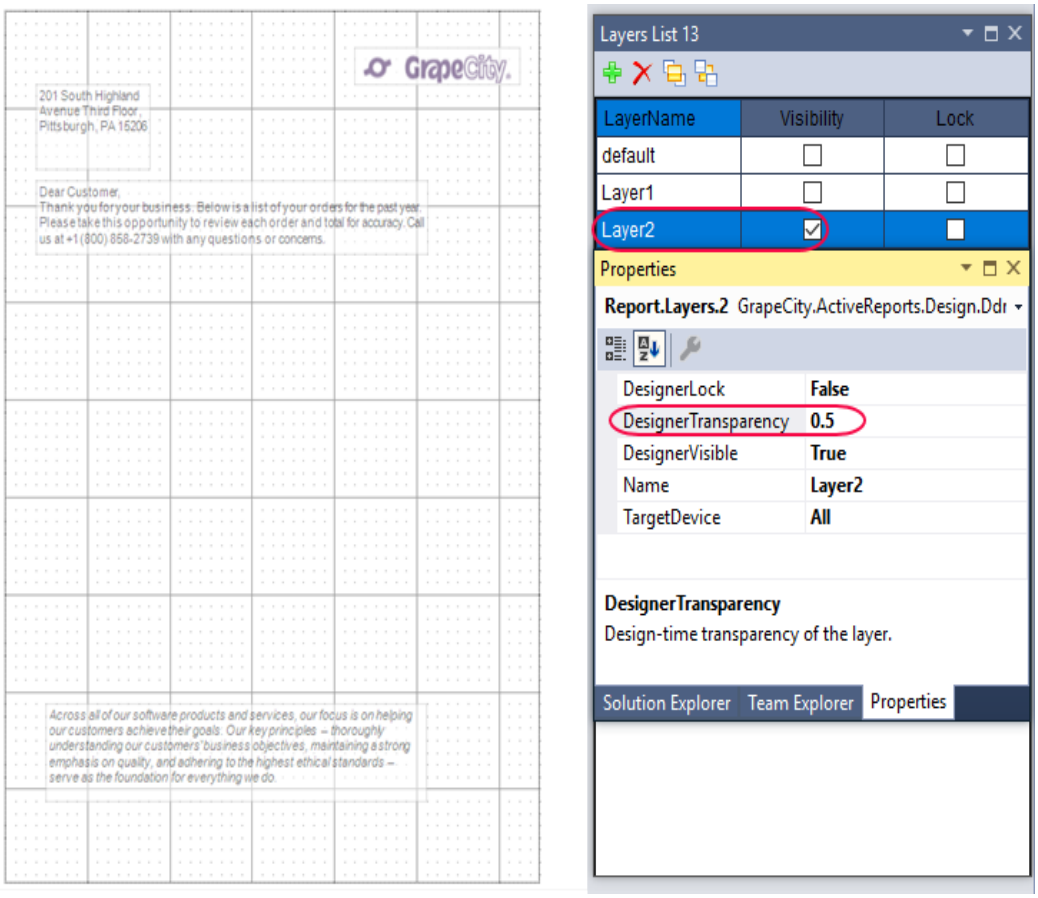
Illustrative Example

Place the scanned image of the letter that needs to be replicated on Layer1 and set the **DesignerLock** property of this Layer to True to make sure the image being traced is not modified or changed by mistake. Please see Working with Layers for details on how to lock a Layer.

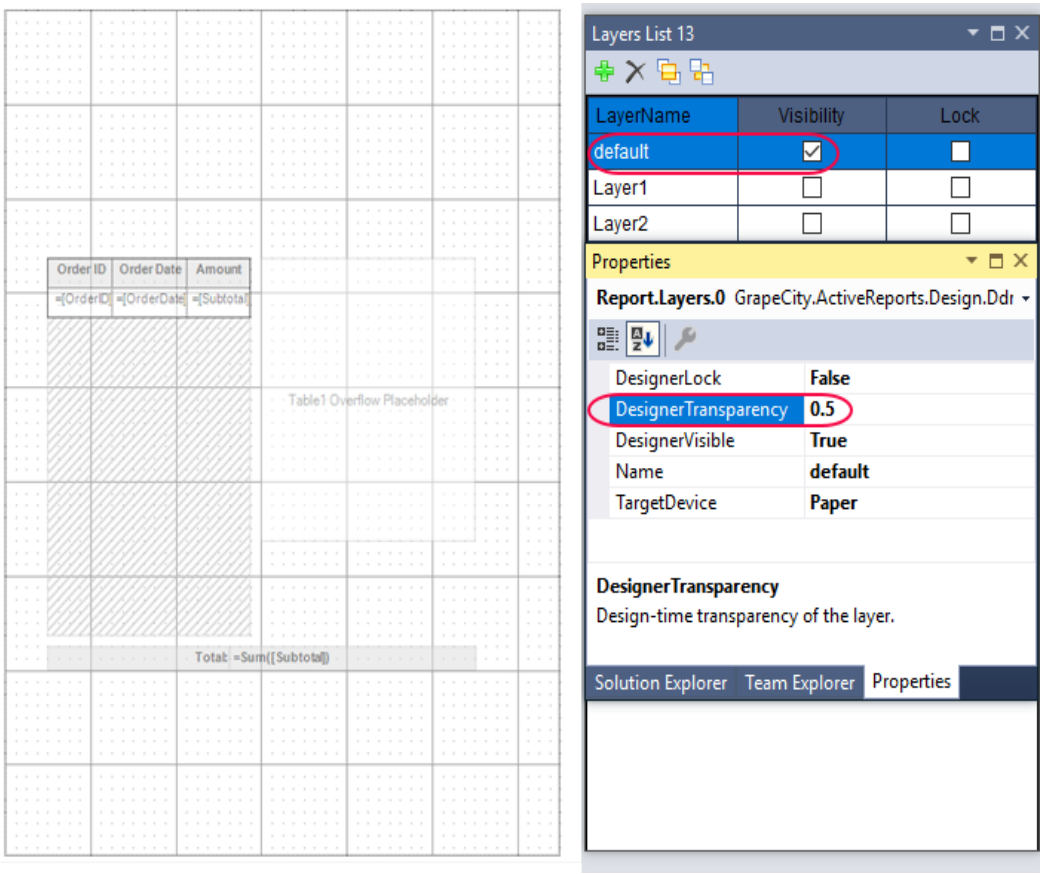




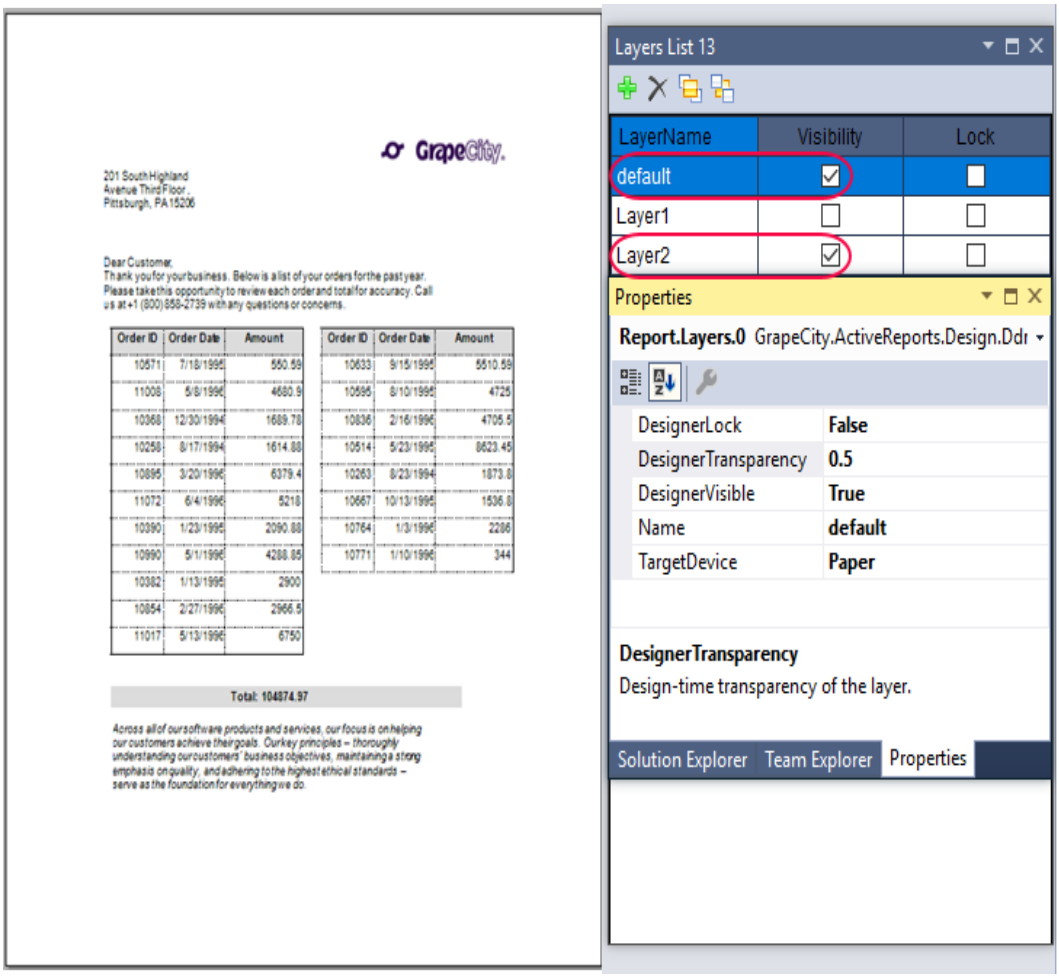
While designing any report, it is advisable to separate the layout, data and logic of your report. In this example, we have placed all the static labels like the logo, header, footer on Layer2 and data bound fields on the Default Layer. This is particularly useful when designing complex report layouts such as tax forms, regulatory notices or bill of lading forms. Working on different Layers makes report designing easier as you can modify one aspect of the report, such as static labels or bound data fields without modifying the entire report layout.



On the Default Layer, place the data bound fields like Order ID, Order Date and Amount. Notice the **DesignerTransparency** of Layer2 above and the Default Layer below is set to 0.5 to display the scanned image placed on Layer1 in the background.



Once the layout being used to copy the layout (scanned image in Layer1) is no longer required, set the **DesignerVisible** property of Layer1 to False to hide the Layer or you can also delete this Layer. DesignerVisible property helps in checking the accuracy of the layout by quickly showing or hiding the controls on the selected layer. In this case we have hidden the visibility of Layer1 to verify the layout of the final output.



In this scenario, Layers are used to replicate the layout of a scanned image and also separate the layout and the data to help organize our report better.

Working with Layers

When you begin working with Layers, access the Layers List window to handle basic functions like adding or removing a Layer from a report at design time. You may also change the settings for each Layer at design time through a set of in-built properties.

Using the Layers List

The Layers List window displays a list of Layers in the report along with their visibility and lock options. You can also add or remove Layers and even send the Layer back or bring it to the front in the Layers List.

In the Layers List window,

- A **Default Layer** is automatically added when you create a new page report. This Layer cannot be deleted or renamed.
- Any Layer can be set as an **Active Layer** in the report by selecting it in the Layers List 14 window. There is only one Active Layer at any given time in a report.

> **Note:** Modifications can only be made to the Active Layer in the report. No modifications are possible on the inactive Layers.

Show or Hide the Layers List

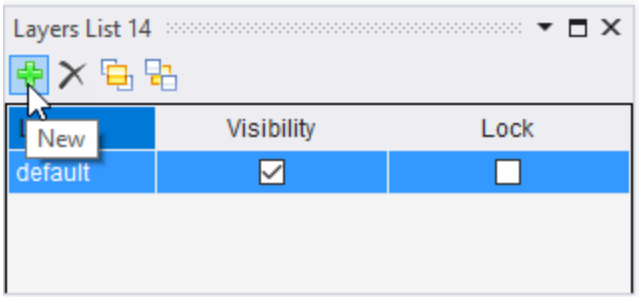
When ActiveReports is installed on your system, a Layers List button is automatically added to the Visual Studio toolbar and it appears every time you create a new application.

1. Right-click the Visual Studio toolbar and select **ActiveReports 14** to display the report designer toolbar. See Toolbar for further details.
2. On the report designer toolbar, click the **View Layers List** button. The **Layers List 14** window appears.
3. Click the **View Layers List** button again to hide the **Layers List 14** window.

> **Note:**
>
> - In case the Layers List window does not appear automatically in your application, select **View > Other Windows > Layers List 14** in Visual Studio.
> - A LayerList control is also available in the Visual Studio toolbar and can be used to add the Layers feature in the End User Designer application. See how to add the Layer List control in the walkthrough on Creating a Basic End User Report Designer (Pro Edition).
> - The stand-alone designer application (GrapeCity.ActiveReports.Designer.exe) also contains a Layers List window. See Stand-alone Designer and Viewer for more information.

Add a Layer

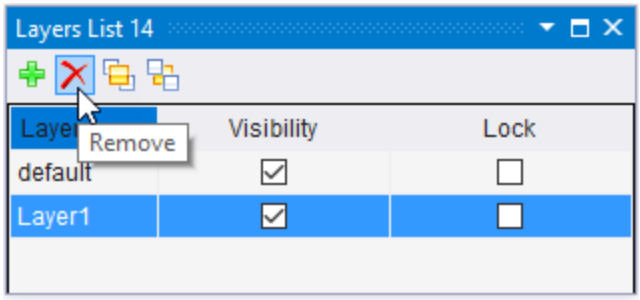
Once a report is created, a Default Layer is automatically added in the Layers List.



1. In the report, select the page on which the Layer is to be added.
2. On the Layers List toolbar, click the **New** button.
3. A new Layer with the name 'Layer1' gets added to the report and the Layers List.

Remove a Layer

All Layers, except the Default Layer, can be removed.

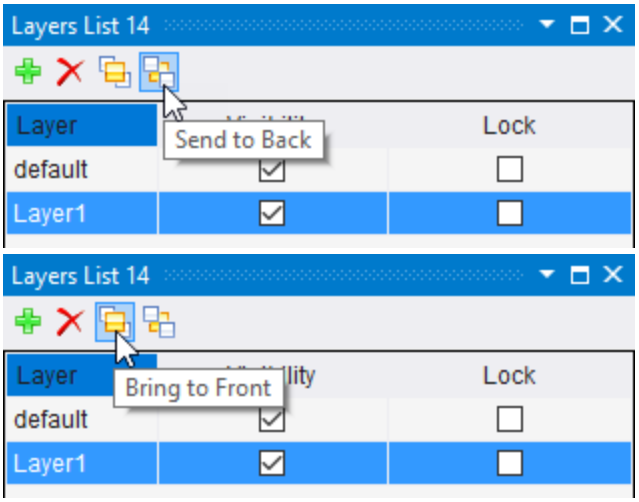


1. In the Layers List, select the Layer to be removed.
2. On the Layers List toolbar, click the **Remove** button to remove the selected Layer.

This removes the selected Layer along with the controls placed on it from the report and the Layers List.

Send to Back/Bring to Front

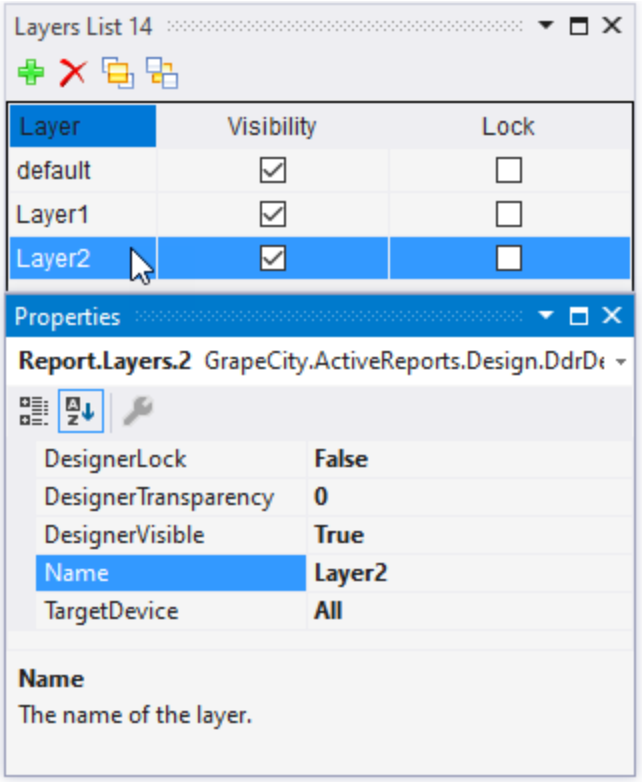
Use the **Send to Back** or **Bring to Front** buttons to send a group of controls placed on a selected Layer to the front or to the back of the controls on other layers.



1. In the Layers List, select the Layer for which the order is to be set.
2. In the Layers List toolbar, click the **Bring to Front** or **Send to Back** button to send the controls placed on a Layer to the front or the back.

Using the Layers Properties

Select a Layer from the Layers List to access the following properties in the property grid.

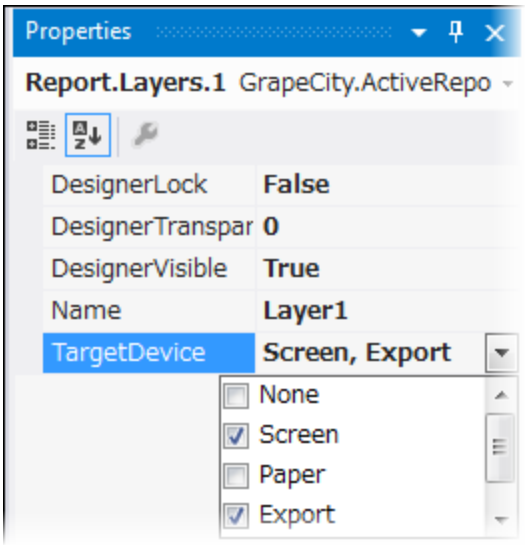


Property	Value	Description
DesignerLock	**True/False**	Locks or unlocks controls placed on a Layer. You cannot move or resize the controls placed on the design surface of a locked Layer through a keyboard or a mouse. Other editing functions like cut, copy or paste and addition or deletion of controls are possible. This property can also be set using the check-box for **Lock** in the Layers List.
DesignerTransparency	**0 to 1**	Sets the transparency of the controls on a Layer at design time to a value between 0 and 1. A Layer with transparency set to 1 is not visible on the designer.
DesignerVisible	**True/False**	Determines if the controls placed on a Layer are visible on the designer or not. This property can also be set using the check-box for **Visibility** in the Layers List.
Name	**Layer Name (string)**	Sets the name of a Layer (except the Default Layer).
TargetDevice	**None, Screen, Paper, Export, All**	Specifies or limits the visibility of controls placed on a Layer based on the selected target. • None: Layer is not visible on any target device • Screen: Layer is visible on the viewers • Paper: Layer is visible on printing

Property	Value	Description
		• Export: Layer is visible on export • All: Layer is visible on all targets i.e. Screen, Paper and Export See View, Export or Print Layers for information on TargetDevice specific outputs.

View, Export or Print Layers

The **GrapeCity.ActiveReports.PageReportModel.Layer.TargetDevice** property determines whether you can view, export or print the controls placed on a Layer. It allows you to show or hide the controls that belong to a Layer on a specific target device. For example, if you want to display controls placed on Layer1 in the WinViewer and export the output to PDF, you can select the **Screen** and **Export** options of the TargetDevice property simultaneously.



Note: TargetDevice property only determines the target (i.e. Screen, Export or Print) where the group of controls placed on a Layer appear. In order to get the output on a viewer or export or print the controls, you need to add code for exporting or printing or adding a viewer to the application.

TargetDevice property allows you to select from the following options:

- **None**
- **Screen**
- **Paper**
- **Export**
- **All**

Setting the TargetDevice Property

Use the following steps to set or change the **TargetDevice** property of a Layer:

1. Select a Layer from the Layers List window. This becomes the Active Layer of the control.
2. In the Properties window, select the **TargetDevice** property.
3. From the dropdown, select out of the options: **None, Screen, Paper, Export** or **All**.
   You can also select more than one option out of Screen, Paper and Export at the same time.

TargetDevice	Output Type	Description
None	-	Controls placed on a Layer cannot be viewed, exported or printed.
Screen	WinViewer	Controls placed on a Layer can be viewed on any of the supported viewers.
	WebViewer	
	WPF Viewer	
Paper	Physical/ Virtual Printer	Controls placed on a Layer are can be printed to physical or virtual printers.
Export	RenderingExtensions - HTML, PDF, Word, Image, Xml, Excel	Controls placed on a Layer are exported to any of the supported file formats.
	Export Filters - HTML, PDF, Tiff, Text, Rft, Excel	
All	All Outputs	Controls placed on a Layer can be viewed, exported and printed.

Tracing Layers

In Page Reports and Rdl reports, you can trace the layout of a pre-printed form to a pixel perfect accuracy using Layers.

This walkthrough illustrates how you can trace data fields accurately from a scanned image of a boarding pass. These steps show the results on a Page Report but are applicable to Rdl Reports also.

Note: Right-click and save the image below in your project folder before starting the walkthrough.

The walkthrough is split up into the following activities:

- Adding an ActiveReport to a Visual Studio project
- Connecting the report to an XML data source

- Adding a dataset
- Creating a report layout on different layers
- Viewing the report

When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout

Run-Time Layout

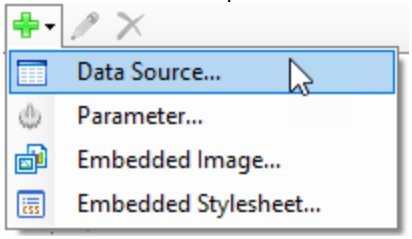
Adding an ActiveReport to a Visual Studio project

1. In Visual Studio, create a new **Windows Forms Application** project.
2. In the Solution Explorer window, right-click Form1, and rename it to **MainForm**.
3. From the **Project** menu, select **Add New Item**.
4. In the Add New Item dialog that appears, select **ActiveReports 14 Page Report** and in the Name field, rename the file to **Boarding**.
5. Click the **Add** button to open a new Page Report in the designer.

See Quick Start for information on adding different report layouts.

Connecting the report to an XML data source

1. From the Solution Explorer, open Boarding.rdlx created earlier.
2. In the Report Explorer, right-click the Data Sources node and select **Add Data Source** or select **Data Source** from the Add button dropdown.



3. In the Report Data Source Dialog that appears, select the **General** page and in the Name field, enter a name like **CustomDS**.

4. In the Type drop down to select a data provider, choose **Xml Provider**.
5. In the connection string section, add the following custom data to your project.

Xml

```
 XmlData =
<Flight ID="31">
  <Passengers>
    <Passenger>
      <PassengerName>Maria Anders</PassengerName>
      <Seat>23A</Seat>
      <Tracking>2322- 030-0074321</Tracking>
    </Passenger>
    <Passenger>
      <PassengerName>Ana Trujillo</PassengerName>
      <Seat>18E</Seat>
      <Tracking>2322- 030-0074343</Tracking>
    </Passenger>
    <Passenger>
      <PassengerName>Antonio Moreno</PassengerName>
      <Seat>25A</Seat>
      <Tracking>6722- 034-6784349</Tracking>
    </Passenger>
    <Passenger>
      <PassengerName>Christina Berglund</PassengerName>
      <Seat>14B</Seat>
      <Tracking>5678- 543-6784349</Tracking>
    </Passenger>
    <Passenger>
      <PassengerName>Thomas Hardy</PassengerName>
      <Seat>28F</Seat>
      <Tracking>9834- 413-6784569</Tracking>
    </Passenger>
  </Passengers>
</Flight>
```

6. Click **OK** to close the dialog box.

Adding a dataset

1. In the Report Explorer, right-click the data source created in the previous step and select **Add Data Set** or select **Data Set** option from the Add button dropdown.
2. In the DataSet Dialog that appears, select the **General** page and enter the name of the dataset as **BoardingData**.
3. On the **Query** page, select **Text** under **Command Type** and enter the following XML path into the **Query** text box to access the data of each passenger.
   `//Passenger`
4. On the **Fields** page, enter the values from the table below to create fields for your report. Values for XML data fields must be a valid XPath expression.

Field Name	Type	Value
Tracking	Database Field	Tracking

PassengerName	Database Field	PassengerName
Seat	Database Field	Seat

5. Click **OK** to close the dialog. Your data set and queried fields appear as nodes in the Report Explorer.

See Adding a DataSet for more information on adding dataset to a data source.

Creating a report layout on different layers

1. In the LayerList window, click the  icon to add a new Layer to the report. Notice that a default Layer is already added in the Layer List.
2. Select Layer1 to make it the Active Layer, and from the Properties window, change the Layer **Name** to **ImgLayer**.
3. In the Layer List window, check the **Lock** check-box to lock **ImgLayer** to make sure the image being traced is not modified or changed by mistake.
4. In the Layer List window,click the **Send To Back** button to move ImgLayer behind the Default Layer.
5. In the LayerList window, select Default to make it the Active Layer.
6. From the Properties window, set the **DesignerTransparency** of the Default Layer to 0.5 to display the scanned image to be placed on **ImgLayer** in the background.
7. From the toolbox, drag a List data region onto the design surface of the report.
8. In the Properties window of the List data region set the following values.

Property Name	Values
DataSetName	BoardingData
FixedSize	Width: 6.27in Height: 3in
Location	Left: 0in Top: 0.125in
Size	Width: 6.27in Height: 2.87in

9. In the Report Explorer window, right-click the **Embedded Images** node and select **Add Embedded Image**.
10. In the **Open** dialog box that appears, browse to the project folder location and select the BoardingPass.jpg image you saved earlier.
11. Click **Open** to embed the image in your project.
12. From the Report Explorer, drag the embedded **BoardingPass** image onto the List data region and set the properties as described in the following table.

Property Name	Property Value
LayerName	ImgLayer
Location	0in, 0.125in
Size	6.27in, 2.875in

13. From the Report Explorer, drag the following fields onto the List data region and set the following properties.

Field Name	Property Values
PassengerName	LayerName: Default Location: 2.91in, 0.35in Size: 1.375in, 0.25in

PassengerName	LayerName: Default Location: 0.15in, 0.9in Size: 1.37in, 0.25in
PassengerName	LayerName: Default Location: 4.5in, 0.9in Size: 1.25in, 0.25in
Tracking	LayerName: Default Location: 0.15in, 2.19in Size: 1.5in, 0.25in
Seat	LayerName: Default Location: 4.5in, 1.45in Size: 1in, 0.375in Font > FontSize: 22pt

14. In the Layer List window, select the Default Layer and then the ImgLayer and change their **TargetDevice** property to **Screen** from the Properties window. See View, Export or Print Layers for more information.

Viewing the report

- Click the preview tab to view the report.

OR

- See Windows Forms Viewer to display report in the Viewer at run time.

Go to Top

Report Appearance

Customize the appearance of reports by applying styles and themes, as described in following topics:

- Styles
- Themes

Styles

What are Styles?

Styles are a set of properties that you can apply to selected controls in your Page or RDL reports to quickly change their appearance. A single style can define properties for font, background color, line spacing, border color, padding, and many more. You can create four different types of styles, namely **Common, Text, Table Of Contents** and **Table Of Contents Level**. For more information on the type of styles and how they differ from each other, see **Working with Styles**.

What are Style Sheets?

ActiveReports provides you with the ability to create multiple styles and store them in a style sheet. A style sheet can be understood as a collection of styles. You can add the style sheet to your Page or RDL reports using the GrapeCity.ActiveReports.PageReportModel.Report.**StyleSheetSource** and GrapeCity.ActiveReports.PageReportModel.Report.**StyleSheetValue** properties and apply the styles to selected controls using the

GrapeCity.ActiveReports.PageReportModel.Style.**StyleName** property. You can either embed the style sheet within your report or save it externally in the ***.rdlx-styles** format. For more information, see **Working with Styles**.

There are two ways to use these style sheets:

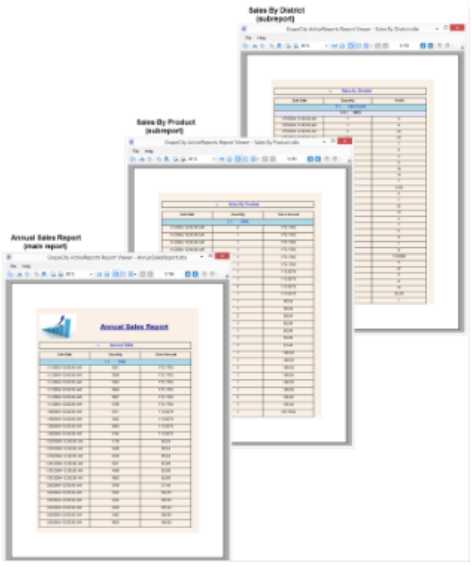
- Embed the style sheets within the report and use its styles on controls in that report
- Save the style sheets externally in ***.rdlx-styles** format and use it in multiple reports

Why use Styles?

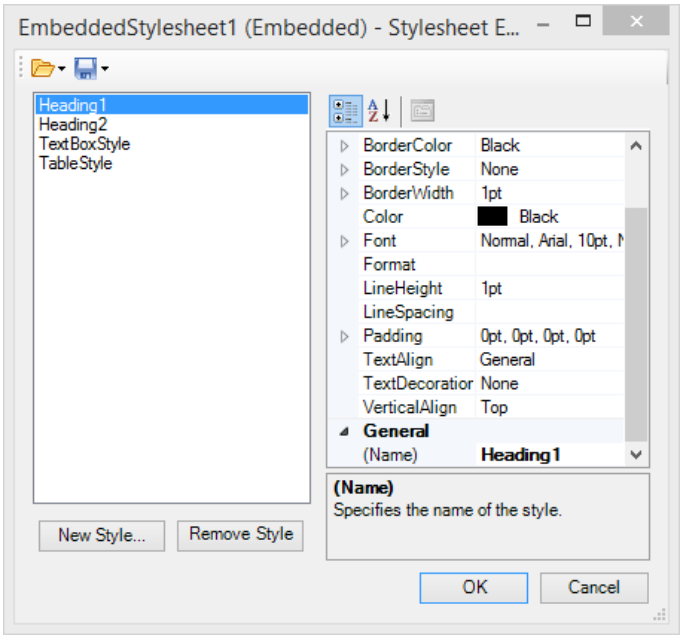
Using styles gives you more control over how you format the report. Let us look at a few scenarios to understand how styles are helpful while designing reports.

Reusing Styles

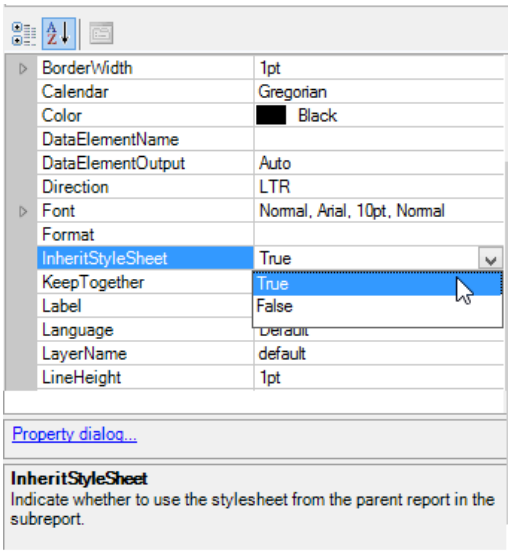
An organization wants to create an Annual Sales Report that consists of multiple subreports, representing Sales by District and Sales by Product. Since all the subreports are a part of the Annual Sales Report, the formatting needs to be consistent. You normally have to manually set properties for each control on the report to format it and then replicate those same set of properties for the two subreports. This can be time consuming and can lead to inconsistent styling in your reports. Let us see how the ActiveReports **Style** feature can help you generate consistent styles in all reports similar to the screenshot below.



Report author can create a style sheet for designing the Annual Sales Report (main report), add styles to the style sheet using the **Stylesheet Editor** dialog. After creating the styles, these styles can be applied to various controls on Annual Sales Report using the **GrapeCity.ActiveReports.PageReportModel.Style.StyleName** property. For more information, see **Working with Styles**.



Once the Annual Sales Report has been designed, the **GrapeCity.ActiveReports.PageReportModel.Subreport.InheritStyleSheet** property can be set to True (by default) for each subreport.



By setting the InheritStyleSheet property to **True**, the style sheet used for the Annual Sales Report is automatically inherited in the subreports. This makes all the styles in the style sheet available to the two subreports. To apply the styles to the report controls in a subreport, you simply need to select the report control and specify the name of the style you want to use in the **StyleName** property.

In this example we can see how styles can help save time and maintain consistent formatting by giving you the flexibility to use the same style sheet in multiple subreports.

You can also use the same style sheet in multiple reports by saving the style sheets externally in ***.rdlx-styles** format. For more information on how to work with external style sheets, see **Working with Styles**.

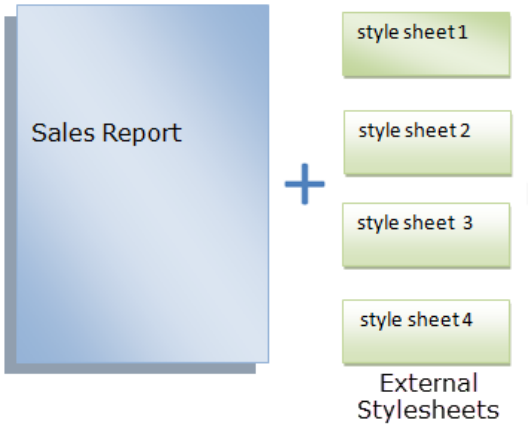
Enhancing Report Portability

In ActiveReports, you can embed external style sheets in a report. This is particularly useful when you want to send reports that are styled using multiple style sheets. Let us take an example of a Sales Report to see how embedded style sheets can help improve the report portability.

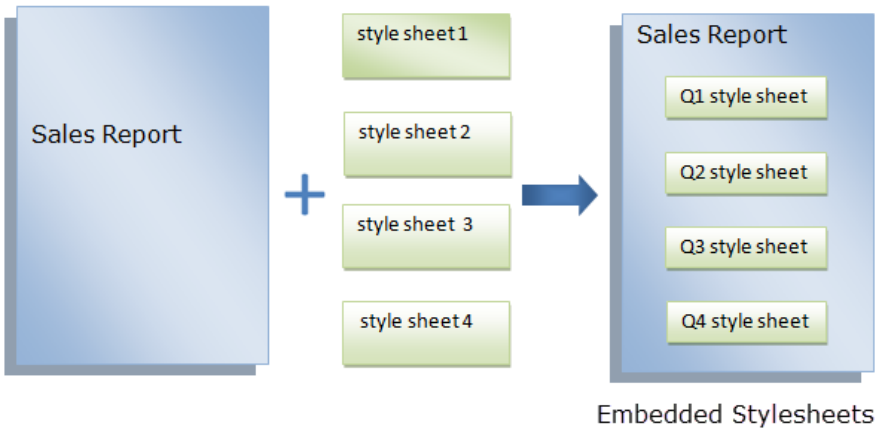
Scenario

An organization wants to send a Sales report that is styled using 4 different external style sheets. While sending the styled report, 5 files have to be sent together, i.e. one report and 4 external style sheets. The person receiving these files needs to maintain and store 5 different files. Moreover, if the location of a style

sheet is changed from the path set in the report, the style sheet will no longer be applied to the report until the path is modified in the report.



By embedding the external style sheets within the Sales report, only 1 file needs to be sent which in turn improves the portability of the report. For more information on how to embed external style sheets, see **Embed External style sheet into a report**



ActiveReports provides you the ability to create styles and store them in a style sheet. You can add these style sheets to your Page or RDL reports and apply the styles to selected controls using the **GrapeCity.ActiveReports.PageReportModel.Style.StyleName** property. You can also save these style sheets on your system.

The styles feature consists of the following elements.

- **Style Sheet Properties**
- **Stylesheet Editor Dialog**
- **Add New Style Dialog**
- **Embed Stylesheet Dialog**
- **Open Embedded Stylesheet Dialog**

Here is some guidance on how to work with style sheets

- **Working with Styles within a Style Sheet**
- **Working with Embedded Style Sheets**
- **Working with External Style Sheets**
- **Applying Styles Through Code**

Style Sheet Properties

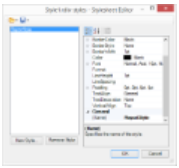
Define the style sheet of a report in the Properties window using the **Source** property and the **Value** property. For subreports, use the InheritStyleSheet property.

Property Name	Description
Source	The source of a report's style sheet. You can choose from the following options: **External** - Choose this option if the style sheet (***.rdlx-styles** format) is located as an external source, such as a local file, an http location or a custom resource. To learn how to create external style sheets, see **Working with External Style Sheets**.

	Embedded - Choose this option if style sheet is embedded in the report. The embedded style sheets are displayed under the **Embedded StyleSheets** node of the Report Explorer.To learn how to create embedded style sheets, see **Working with Embedded Style Sheets**.
Value	The style sheet to apply to the report. You can choose from the following options: **Expression** - Opens the Expression Editor dialog to create a valid expression. **New** - Opens the **New Stylesheet Editor** dialog to create an external or embedded style sheet. **Open file** - Opens the **Open Stylesheet from file** dialog to navigate to a local style sheet file. This option is only available for external style sheets. For embedded style sheets, a list of available style sheets in the report is provided.
InheritStyleSheet	The style sheet to inherit in a subreport. Setting the **InheritStyleSheet** property to True (default value) inherits the style sheet of the main report in the subreport.

Note: Field values are not evaluated when used as an expression in Stylesheet Value, StyleName and Styles properties.

Stylesheet Editor Dialog



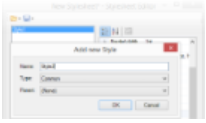
You can open the **Stylesheet Editor** dialog by selecting the **Stylesheet Editor** option from the **Report** menu of the stand-alone designer or Visual Studio .NET designer.

The Stylesheet Editor dialog consists of the following elements.

Elements	Description
Open Stylesheet from File	Opens a style sheet (***.rdlx-styles** format) located externally.
Open Embedded Stylesheet	Opens a style sheet embedded in the report.
Save Stylesheet to File	Saves the current style sheet as an external style sheet in ***.rdlx-styles** format.
Embed Stylesheet	Embeds the current style sheet in the report.
New Style	Creates a new style in the current style sheet.
Remove Style	Removes a style from the current style sheet.
Property window	Modifies the properties of the selected style based on the selected style type. Available style properties change depending on the type of style selected. You set the style type when you create a new style. The style type is selected when a new style is created.
OK	Saves the current style.
Cancel	Closes the dialog without saving the changes.

Note: The values set in the Properties window override the values defined in the report's style sheet. The overridden values are displayed in bold in the Properties window.

Add New Style Dialog



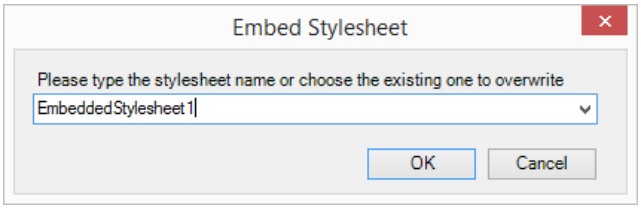
You can open the **Add New Style** dialog by clicking the **New Style** option in the **Stylesheet Editor** dialog.

The **Add New Style** dialog consists of the following elements.

Elements	Description
Name	Contains the name of the new style.

Type	Sets the type of control to which you can apply the style, which determines the options that are available in the **Properties window** of the **Stylesheet Editor** dialog. Common Apply this style type to the following report controls: • CheckBox • Image • List • Tablix • Shape • Table • TableOfContents • TextBox Text Apply this style type to the **TextBox** report control. It includes all properties of the Common style type, plus it offers properties specific to the TextBox control. TOC Apply this style type to the **TableOfContents** control. TOC Level Apply this style type to the **ToC.Level** object of the TableOfContents control.
Parent	Represents the parent style of a new style. If the parent style is specified, the property values are taken from the selected parent style values. By default, the parent style is set to **None**.

Embed Stylesheet Dialog

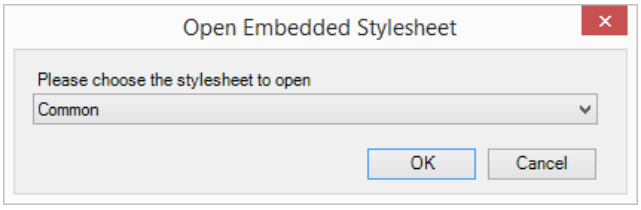


You can access the **Embed Stylesheet** dialog by selecting the **Save current Stylesheet** option and then selecting **Embed Stylesheet** in the **Stylesheet Editor** dialog.

The **Embed Stylesheet** dialog consists of the following elements.

Elements	Description
Drop-down list box for style sheet name	Enter a name for the embedded style sheet, or choose an existing style sheet from the drop-down list box to overwrite.

Open Embedded Stylesheet Dialog



You can access the **Open Embedded Stylesheet** dialog by selecting the **Open stylesheet** option and then selecting **Open embedded Stylesheet** from the **Stylesheet Editor** dialog.

The Open Embedded Stylesheet dialog consists of the following elements.

Elements	Description
Drop-down list box for opening style sheet	Provides a drop-down list box to choose an existing style sheet to overwrite. You can also enter a new name for the style sheet here.

Here is some guidance on how to work with style sheets

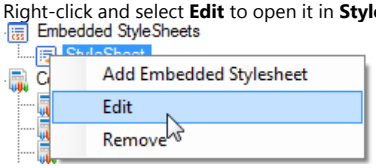
- **Working with Styles within a Style Sheet**
- **Working with Embedded Style Sheets**
- **Working with External Style Sheets**
- **Applying Styles Through Code**

Working with Styles within a Style Sheet

For any of these operations, you first need to open the style sheet in the editor.

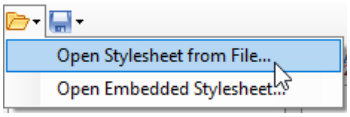
To open the editor for embedded style sheets

1. In the Report Explorer, expand the **Embedded StyleSheets** node and select the existing style sheet you want to edit.
2. Right-click and select **Edit** to open it in **Stylesheet Editor** dialog.



To open the editor for external style sheets

1. In the stand-alone designer or Visual Studio designer, click the Report menu and select the **Stylesheet Editor**.
2. In the Stylesheet Editor dialog, click the Open button and select the **Open Stylesheet from File** option.



3. In the **Open** dialog, navigate to the ***.rdlx-styles** file that you want to open.
4. Click **Open** to open the external stylesheet in the **Stylesheet Editor**.

To add a new style to a style sheet

1. In the **Stylesheet Editor** dialog, click the **New Style** button to add a new style.
2. In the **Add New Style dialog**, enter the **Name** of the style, and then select the **Type** and **Parent** style.

> **Tip**: For more information on the style types, see **Working with Styles**. To create style types for TableOfContents controls and heading levels, see Apply styles to the TableOfContents control and Apply styles to the TableOfContents levels.

To modify a style in a style sheet

1. In the **Stylesheet Editor**, select the existing style that you want to modify and use the property fields on the right to make the changes.
2. Click **OK** to save the changes.

To remove a style from a style sheet

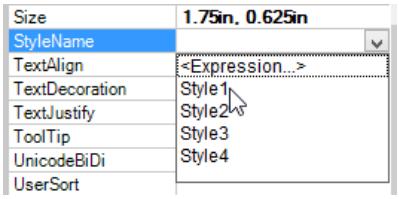
1. In the **Stylesheet Editor**, select the style that you want to remove and click **Remove Style.**
2. Click **OK** to save the changes.

To use a style from a style sheet at design time

1. Click the gray area around the report to select it, and under the Properties window, click the **Property dialog** link in the Commands section. See Properties Window for more information on how to access commands.
2. In the Report dialog, go to the **Appearance** page.
3. In the **Appearance** page, set the Stylesheet Source to Embedded and in the Value field select an existing embedded style sheet. (Or select External and select the <Open File> option and navigate to an ***.rdlx-styles** external style sheet.)

> **Tip**: You can also access the **Source** and **Value** properties in the Properties window by expanding the **Stylesheet** node. For more details on the **Source** and **Value** properties, see **Working with Styles**.

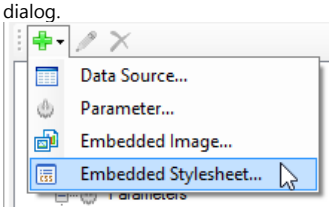
4. Click **OK** to close the dialog.
5. On the design surface, select the control you want to apply the style to.
6. In the Properties Window, from the **StyleName** property drop-down, select a style to apply to the controls.



Working with Embedded Style Sheets

Create and save a style sheet

1. In the Report Explorer, right-click the **Embedded StyleSheets** node, and select the **Add Embedded Stylesheet** option to access the **Stylesheet Editor** dialog.



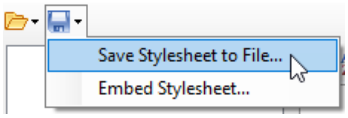
> **Tip**: You can also access the **Stylesheet Editor** dialog from the Report Explorer by clicking the Add button and selecting **Embedded Stylesheet**. In the stand-alone designer or Visual Studio designer, from the **Report** menu, select **Stylesheet** Editor.

2. Click the Save button and select **Embed Stylesheet** to embed the style sheet into the report.
3. Enter a name for the style sheet or choose an existing style sheet from the drop-down to overwrite, and then click **OK** to save the embedded style sheet.

All the saved style sheets embedded in the report appear under the **Embedded StyleSheets** node in the Report Explorer.

Save embedded style sheet as an external style sheet

1. In the Report Explorer, expand the **Embedded StyleSheets** node and select the embedded style sheet.
2. Right-click and select **Edit** to open the **Stylesheet Editor** dialog.
3. In the Stylesheet Editor dialog, click the Open button and select the **Save Stylesheet to file** option to save the embedded style sheet externally.



4. In the **Save As** dialog, navigate to the location where you want to save the style sheet, provide a name for the style sheet and click the **Save** button to save it as an external ***.rdlx-styles** file.

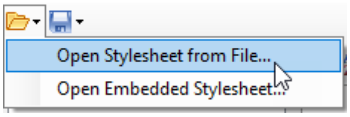
Working with External Style Sheets

Create and save a style sheet

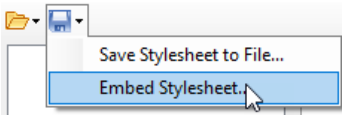
1. In the stand-alone designer or Visual Studio designer, click the **Report** menu and select the **Stylesheet Editor**.
2. In the Stylesheet Editor dialog, click the Open button and select the **Save Stylesheet to file** option.
3. In the **Save As** dialog, navigate to the location where you want to save the style sheet, provide a name for the style sheet and click the **Save** button to save it as an external ***.rdlx-styles** file.

Embed External style sheet into a report

1. In the stand-alone designer or Visual Studio .NET designer, click the Report menu and select the **Stylesheet Editor**.
2. In the Stylesheet Editor dialog, click the Open button and select the **Open Stylesheet from file** option.



3. In the Open dialog, navigate to the external style sheet **(*.rdlx-styles file)** that you want to load and click the **Open** button to load it in the Stylesheet Editor dialog.
4. In the Stylesheet Editor dialog, click the Save button and then select the **Embed Stylesheet** option.



5. In the **Embedded Stylesheet** dialog, enter a name for the style sheet and then click **OK** to embed the loaded style sheet into your report.

All the saved style sheets embedded in the report appear under the **Embedded StyleSheets** node in the Report Explorer.

Applying Styles Through Code

1. In Visual Studio, create a new **Page Report Application** or open an existing one.
2. On the Form.cs or Form.vb that opens, double-click the title bar to create the Form_Load event.
3. Add the following code inside the Form_Load event.

Visual Basic

Visual Basic.NET code. Paste INSIDE the Form Load event.

```
'Path and Name of the loaded PageReport
Dim filePath As String = "C:\SampleReport.rdlx"
Dim pageReport As New GrapeCity.ActiveReports.PageReport(New System.IO.FileInfo(filePath))
Dim reportDocument As New GrapeCity.ActiveReports.Document.PageDocument(pageReport)

' Set the style sheet source and value using external style sheets
reportDocument.PageReport.Report.StyleSheetSource =
GrapeCity.ActiveReports.PageReportModel.StyleSheetSource.External
reportDocument.PageReport.Report.StyleSheetValue = "C:\ExternalStyle.rdlx-styles"

' Set the style sheet source and value using embedded style sheets
reportDocument.PageReport.Report.StyleSheetSource =
GrapeCity.ActiveReports.PageReportModel.StyleSheetSource.Embedded
reportDocument.PageReport.Report.StyleSheetValue = "EmbeddedStylesheet1

' Add a Textbox control and apply style
Dim text As New GrapeCity.ActiveReports.PageReportModel.TextBox()
text.Value = "Sample Text"
text.Style.StyleName = "Style1"
pageReport.Report.Body.ReportItems.Add(text)
viewer1.LoadDocument(reportDocument)
```

C#

C# code. Paste INSIDE the Form Load event.

```
//Path and Name of the loaded PageReport
string filePath = @"C:\SampleReport.rdlx";
GrapeCity.ActiveReports.PageReport pageReport =
new GrapeCity.ActiveReports.PageReport(new System.IO.FileInfo(filePath));
GrapeCity.ActiveReports.Document.PageDocument reportDocument =
new GrapeCity.ActiveReports.Document.PageDocument(pageReport);

// Set the style sheet source and value using external style sheets
reportDocument.PageReport.Report.StyleSheetSource =
GrapeCity.ActiveReports.PageReportModel.StyleSheetSource.External;
reportDocument.PageReport.Report.StyleSheetValue = @"C:\ExternalStyle.rdlx-styles";

// Set the style sheet source and value using embedded style sheets
reportDocument.PageReport.Report.StyleSheetSource =
GrapeCity.ActiveReports.PageReportModel.StyleSheetSource.Embedded;
reportDocument.PageReport.Report.StyleSheetValue = "EmbeddedStylesheet1";

// Add a Textbox control and apply style
GrapeCity.ActiveReports.PageReportModel.TextBox text =
new GrapeCity.ActiveReports.PageReportModel.TextBox();
text.Value = "Sample Text";
```

```
text.Style.StyleName = "Style1";
pageReport.Report.Body.ReportItems.Add(text);
viewer1.LoadDocument(reportDocument);
```

Themes

A theme is a collection of properties that defines the appearance of a report. A theme includes colors, fonts, images, and constant expressions that you can apply to report elements once you add a theme to a report.

You can add one or many themes to a report. If a report has multiple themes, you can use the report's **GrapeCity.ActiveReports.PageReportModel.CollateBy** enumeration to control the page order in a report. See **Set Up Collation** for more information.

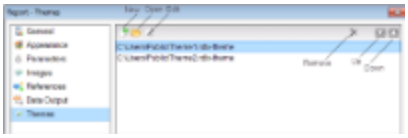
The **Theme Editor** and the **Report - Themes** dialog allow you to manage themes in a report.

In the **Theme Editor**, you can create a new theme by setting colors, fonts, images, and use constant expressions in a theme and then saving a new theme as an .rdlx-theme file on your local machine. Then you can add this theme to your report in the **Report – Themes** dialog. Also, in the **File** menu, select **Open** to open and modify an existing theme and select **Save** or **Save As** to save the changes on your local machine.

To access the Theme Editor

From the **Start** menu, go to **All Programs > GrapeCity > ActiveReports** and select **ActiveReports Theme Editor**.

The **Report – Themes** dialog displays the report's themes. This dialog allows you to create a new theme, add, modify or remove an existing one, as well as rearrange the order of themes if a report has many themes. When you select to create or modify a theme, the **Theme Editor** is opened.



To access the Theme - Report dialog

1. In the Designer, click the gray area around the report page to select a report.
2. Do one of the following:
    - In the Properties window, select the **Themes** property and click the ellipsis (...) button to open the Report - Themes dialog.
    - With the report selected, in the Properties window under properties where the commands are displayed, click the **Property dialog** link. In the Report dialog that appears, go to Themes. See Properties Window for further information on commands.

- On the **Report** menu, select **Report Properties** and go to Themes in the Report dialog that appears.

Create and add themes

A theme is a collection of properties that defines the appearance of a report. A theme includes colors, fonts, images, and expressions that you can apply to report elements once you add a theme to a report.

You can add one or many themes to a report. If a report has multiple themes, you can use the report's **GrapeCity.ActiveReports.PageReportModel.Report.CollateBy** property to control the page order in a report.

Use the following instructions to create and add themes.

To create a new theme

1. From the **Start** menu, go to **All Programs > GrapeCity > ActiveReports** and select **ActiveReports Theme Editor**.
2. In the Theme Editor that opens, define the colors, fonts, images, and constant expressions properties for your new theme under the corresponding tabs.
3. On the **File** menu, select **Save**.
4. Choose a directory on your local machine and enter the name of a new theme, then click **Save**.

To add a theme to the report

1. In the Designer, click the gray area around the report page to select a report.
2. In the Properties window, select the **Themes** property and click the ellipsis (...) button to open the Report - Themes dialog.
3. In the Report - Themes dialog that opens, click the **Open...** icon above the list of themes.
4. In the Open dialog that appears, select a theme file from your local files and click **Open**.

Customize and apply themes

Use the following instructions to customize an existing theme and apply it to your report.

To modify a theme

1. In the Designer, click the gray area around the report page to select a report.
2. In the Properties window, select the **Themes** property and click the ellipsis (...) button to open the Report - Themes dialog.
3. In the Report - Themes dialog that opens, select an existing report theme.
4. Click the **Edit...** icon above the list of themes.
5. In the Theme Editor that opens, modify the theme properties and click **OK** to close the dialog.

To apply a theme color

1. In the Designer, select the report's control (for example, a **TextBox**).
2. In the Properties window, go to the color-related property (for example, the **BackgroundColor** property) and click the arrow to display the drop-down list of values.
3. In the list that appears, go to the **Theme** tab and select the color you want.

To apply a theme font

1. In the Designer, select the report's control (for example, a **TextBox**).
2. In the Properties window, go to a property from the Font properties group (for example, the **Font** property) and click the arrow to display the drop-down list of values.
3. In the values list that appears, select a font defined in a theme (for example, **=Theme.Fonts!MinorFont.Family**).

Use Constant Expressions in a theme

In the Theme Editor, you can define constant expressions to be used in a theme. Later, you can apply a constant expression to the report's control by selecting it in the Value field of that control.

Also, you can apply a constant expression to a report's control in code by using the following syntax (VB code example):

```
=Theme.Constants!Header
=Theme.Constants("Header")
```

Constant expressions allow you to define a name and an associated value to be used in themes.

Use the following instructions to create and use constant expressions in themes.

To define a constant expression

1. In the Theme Editor, go to **Constants**.
2. Double-click the field under **Name** and enter the Constant name (for example, **Header**).
3. In the next field to the right, under **Value**, enter the Constant value (for example, **Invoice#**).

To use a constant expression

1. In the Designer, select the report's control (for example, a **TextBox**).
2. In the Properties Window, go to the **Values** field and select the **<Expression>** option from the drop-down list to open the Expression Editor.
3. In the Expression Editor, expand the **Themes** node with the constant expressions defined in the report theme.
4. In the Themes node, select a constant and then click the **Replace** or **Insert** button.
5. Click **OK** to add the constant expression in the TextBox.

Set Up Collation

You can add multiple themes to a report. In this case, the report renders a combination of multiple outputs for each theme. For example, if a report has two themes, then the report output includes a combination of the first and the second themes, applied to each report page. You can control the combination rules of the report output in the **GrapeCity.ActiveReports.PageReportModel.Report.CollateBy** property.

> ⚠ **Caution:** If you are using collation in a report, you cannot use interactive features, such as drill down, links, document map, and sorting.

You can control the page order of a rendered report with multiple themes by selecting the collation mode in the **CollateBy** property of the report:

> **Note:** The collection of constant expressions must be the same in all themes of a report. See **Use Constant Expressions in a Theme** for further information.

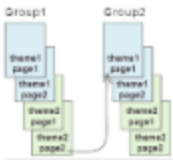
1. In the Designer, click the gray area around the report page to select the report.
2. In the Properties Window, go to the **CollateBy** property and select one of the available options:
    - **Simple**. Renders report pages without any specific sorting. For example, if you have a report with 2 themes, the report renders all pages with theme 1, then all pages with theme 2.

    - **ValueIndex**. Sorts report pages by page number. For example, if you have a report with 2 themes, the report renders page 1 for theme 1 and 2, then page 2 for theme 1 and 2, and so on.

    - **Value**. Sorts report pages by the grouping expression that you specify in the report's FixedPage dialog. For example, if you have a report with 2 themes with grouping, the report renders group1 (pages 1 and 2 of theme1, then pages 1 and 2 of theme2), then group 2 (pages 1 and 2 of theme1, then pages 1 and 2 of theme2), and so on.

      > **Note:** In RDL Reports, the **Value** collation mode is not available by design.

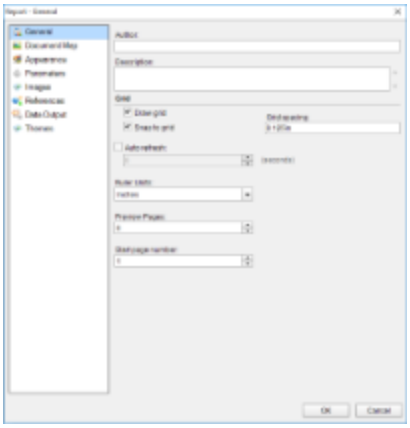
See Add Page Numbers for information on setting cumulative page count formats for Page Report.

Report Dialog

In a page report or a RDL report, you can set the basic report properties in the Report dialog. You can access this dialog by doing one of the following:

- Go to the Visual Studio Report menu and select **Report Properties**.
- In the Report Explorer, right-click the Report node and from the context menu that appears, select **Report Properties**.
- In the Report Explorer, select the Report node and in the Commands section at the bottom of the Properties Window, click the **Property dialog** command.
- Right-click the gray area outside the design surface to select the Report and in the Commands section at the bottom of the Properties Window, click the **Property dialog** command.

The Report dialog provides the following pages where you can set report properties:



General

The General page of the Report dialog allows you to control the following items:

- **Author**: Enter the name of the report author here.
- **Description**: Enter a description of the report here.
- **Draw grid**: Clear this check box to remove the grid lines from the report design surface.
- **Snap to grid**: Clear this check box to allow free placement of report items on the report design surface instead of the automatic alignment of report items with grid lines.
- **Grid spacing**: Enter the spacing between grid lines in inches. The default value is 0.125 inches.
- **Page headers**: These options are enabled when you set a Page Header in a Report Definition Language (RDL) report.
  - **Print on first page** - Select this check box to set the page header on the first page of the report.
  - **Print on last page** - Select this check box to set the page header on the last page of the report.
- **Page footers**: These options are enabled when you set a Page Footer in a RDL report.
  - **Print on first page** - Select this check box to set the page footer on the first page of the report.
  - **Print on last page** - Select this check box to set the page footer on the last page of the report.
- **Auto refresh**: Select this check box to automatically refresh the pages of the report at regular intervals. When this check box is selected, you can supply the interval in seconds.
- **Ruler Units**: Set the ruler units in Inches or Centimeters.
- **Preview Pages**: Set the number of pages to display in the Preview tab. Minimum values is 0 and maximum is 10000 pages. If the value is set to 0, the Preview tab displays all the pages. By default, the Preview tab displays all the pages.

Appearance

The Appearance page of the Report dialog allows you to control the page layout for your report.

Columns

- **Number of columns**: Enter the number of columns you want to use in your report.
- **Spacing**: Enter the number of inches of space to use between columns.

Page Layout

- **Paper Size**: Select one among the standard paper sizes from the dropdown.
- **Width**: Specify the width of the layout.
- **Height**: Specify the height of the layout.
- **Left margin**: Specify the Left margin for the layout.
- **Right margin**: Specify the Right margin for the layout.
- **Top margin**: Specify the Top margin for the layout.
- **Bottom margin**: Specify the Bottom margin for the layout.
- **Orientation**: Select one among **Portrait** and **Landscape** as your page orientation.

Design

- **Output background only at design-time**: Select the checkbox to display background (for example, background image or background color) in the design tab only.

Stylesheet

- **Source**: Select one among **Embedded** and **External** as your style sheet source.
- **Value**: Select the style sheet to apply to the report. Following options are available:

  Expression - opens the Expression Editor dialog to set an expression.

  New - opens the **New Stylesheet Editor** dialog for creating an external or embedded style sheet.

  Open file - opens the **Open Stylesheet from file** dialog for navigating to a local style sheet file. This option is only available for external style sheets.
  In case of **embedded style sheets**, a list of available style sheets in the report is provided.

Parameters

The Parameters page of the Report dialog allows you to control how a user interface is presented to your users for each parameter. See Parameters for further information.

Images

The Images page of the Report dialog allows you to add and modify images for your report. Click the **Open** button located at the top left of the page to display the Open file dialog, where you can navigate to an image. Once you select an image file and click the **Open** button, a thumbnail of the image is displayed in the Image column, and the Name and MIME Type values are automatically populated in their respective columns.

You can use the images you add here in the Image control. The **MIME Type** column provides a combo-box with a list of image file extensions, where you can change default file filter of the added image.

You can also use the **Remove** button located at the top right of the page to remove any added image.

References

The References page of the Report dialog allows you to add references to assemblies and classes so that you can call methods from them in expressions throughout your report. You can also access the References dialog from the Properties Window by selecting the **Classes** (Collection) or **References** (Collection) property of a report and clicking the ellipsis button that appears.

Assembly Name

This is a list of the assemblies available for use in your report. You can delete assemblies using the **Remove** button, or add them using the **Open** button which presents the Open file dialog.

Classes

This is a list of instance-based classes you can create for use in your report.

- **Class name**: Enter the namespace and name of the class here. (i.e. Invoicing.GetDueDate)
- **Instance name**: Enter a name for the instance of the class here. (i.e. m_myGetDueDate)

Data Output

The Data Output page of the Report dialog allows you to control how the report's data is rendered in XML exports.

- **Element name**: Enter the name you want to appear as the top level data element in your exported XML file.
- **Data transform (.xsl file)**: Enter the name of the XSL file you want to use as a style sheet for the exported XML file.
- **Data schema**: Enter the schema or namespace to use for validating data types in the exported XML file.
- **Render textboxes as**: Choose whether to render textboxes as Attributes or Elements in the exported XML file.
  - Attributes example: `<table1 textbox3="Report created on: 7/26/2005 1:13:00 PM">`
  - Elements example: `<table1> <textbox3>Report created on: 7/26/2005 1:13:28 PM</textbox3>`

Themes

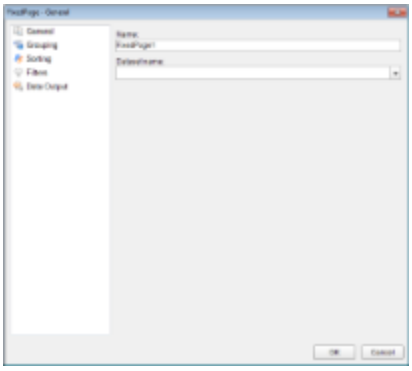
The Themes page of the Report dialog displays the report's themes. This dialog allows you to create a new theme, add, modify or remove an existing one, as well as rearrange the order of themes if a report has many themes. When you select to create or modify a theme, the **Theme Editor** is opened. See Themes for further information.

Fixed Page Dialog

In a Page Report, you can set the basic properties for your page from the FixedPage dialog. You can access the FixedPage dialog by doing one of the following:

- Right-click the gray area outside the design surface and from the context menu that appears, select **Fixed Layout Settings**.
- Click the gray area outside the design surface to select the Report and in the commands section at the bottom of the Properties Window, click the **Fixed Layout Settings** command.
- Click the design surface to select the Page and in the Commands section at the bottom of the Properties Window, click the **Property dialog** command.

- In the Report Explorer, select the Report node or the page node and in the commands section at the bottom of the Properties Window, click the **Fixed Layout Settings** or **Property dialog** command respectively.



The FixedPage dialog provides the following pages where you can set the page properties:

General

The General page of the FixedPage dialog allows you to control the following properties:

- **Name**: Enter a name for the Page report. This name is used to call the page in code so it should be unique within the report.
- **Dataset name**: Select a dataset to associate with the page report. The dropdown list is automatically populated with all the datasets in the report's dataset collection.

Grouping

The Grouping page is useful when you want to show data grouped on a field or an expression on report's pages. See Group Data for more information.

The Grouping page contains following tabs which provide access to various grouping properties:

General

- **Name**: Enter a name for the Fixed Layout group that is unique within the report. This property cannot be set until after a Group on expression is supplied. A name is created automatically if you do not enter one.
- **Group On**: Enter an expression to use for grouping the data.
- **Document map label**: Enter an expression to use as a label to represent this item in the table of contents (document map).

Filters

The Filters tab allows you to add and control the Filter collection for the Fixed Layout Group. Use the + button to add a filter and the X button to delete a filter. You need to provide three values to add a new filter to the collection:

- **Expression**: Enter the expression to use for evaluating whether data should be included in the group.
- **Operator**: Select from the following operators to decide how to compare the expression to the left with the value to the right:
  - **Equal** Only choose data for which the value on the left is equal to the value on the right.
  - **Like** Only choose data for which the value on the left is similar to the value on the right. For more information on using the **Like** operator, see the MSDN Web site.
  - **NotEqual** Only choose data for which the value on the left is not equal to the value on the right.

    - **GreaterThan** Only choose data for which the value on the left is greater than the value on the right.
    - **GreaterThanOrEqual** Only choose data for which the value on the left is greater than or equal to the value on the right.
    - **LessThan** Only choose data for which the value on the left is less than the value on the right.
    - **LessThanOrEqual** Only choose data for which the value on the left is less than or equal to the value on the right.
    - **TopN** Only choose items from the value on the left which are the top number specified in the value on the right.
    - **BottomN** Only choose items from the value on the left which are the bottom number specified in the value on the right.
    - **TopPercent** Only choose items from the value on the left which are the top percent specified in the value on the right.
    - **BottomPercent** Only choose items from the value on the left which are the bottom percent specified in the value on the right.
    - **In** Only choose items from the value on the left which are in the array of values specified on the right. Selecting this operator enables the Values list at the bottom.
    - **Between** Only choose items from the value on the left which fall between the pair of values you specify on the right. Selecting this operator enables two Value boxes instead of one.
- **Value**: Enter a value to compare with the expression on the left based on the selected operator. For multiple values used with the **Between** operator, the lower two value boxes are enabled.

- **Values**: When you choose the **In** operator, you can enter as many values as you need in this list.

Data Output

The Data Output tab allows you to control the following properties when you export to XML:

- **Element name**: Enter a name to be used in the XML output for this group.
- **Collection**: Enter a name to be used in the XML output for the collection of all instances of this group.
- **Output**: Choose Yes or No to decide whether to include this group in the XML output.

Layout

- **Has own page numbering**: Check this box to enable section page numbering like Page N of M (Section). See Add Page Numbers for further information on setting page numbering in Page Report.

Sorting

The Sorting page of the FixedPage dialog allows you to enter sort expressions for sorting data alphabetically or numerically.

Expression: Enter an expression by which to sort the data.

Direction: Select whether you want to sort the data in an Ascending or Descending order.

Filters

The Filters page of the FixedPage dialog allows you to control the Filter collection for the Fixed Layout. Use the + button to add filters and X buttons to delete them. You need to provide three values to add a new filter to the collection:

- **Expression**: Enter the expression to use for evaluating whether data should be included in the Fixed Layout.
- **Operator**: Select from the following operators to decide how to compare the expression to the left with the value to the right:
    - **Equal** Only choose data for which the value on the left is equal to the value on the right.

  - **Like** Only choose data for which the value on the left is similar to the value on the right. For more information on using the **Like** operator, see the MSDN Web site.
  - **NotEqual** Only choose data for which the value on the left is not equal to the value on the right.
  - **GreaterThan** Only choose data for which the value on the left is greater than the value on the right.
  - **GreaterThanOrEqual** Only choose data for which the value on the left is greater than or equal to the value on the right.
  - **LessThan** Only choose data for which the value on the left is less than the value on the right.
  - **LessThanOrEqual** Only choose data for which the value on the left is less than or equal to the value on the right.
  - **TopN** Only choose items from the value on the left which are the top number specified in the value on the right.
  - **BottomN** Only choose items from the value on the left which are the bottom number specified in the value on the right.
  - **TopPercent** Only choose items from the value on the left which are the top percent specified in the value on the right.
  - **BottomPercent** Only choose items from the value on the left which are the bottom percent specified in the value on the right.
  - **In** Only choose items from the value on the left which are in the array of values specified on the right. Selecting this operator enables the Values list at the bottom.
  - **Between** Only choose items from the value on the left which fall between the pair of values you specify on the right. Selecting this operator enables two Value boxes instead of one.

- **Value**: Enter a value to compare with the expression on the left based on the selected operator. For multiple values used with the **Between** operator, the lower two value boxes are enabled.

- **Values**: When you choose the **In** operator, you can enter as many values as you need in this list.

Data Output

The Data Output page of the FixedPage dialog allows you to control the following properties when you export to XML:

- **Element name**: Enter a name to be used in the XML output for the Page report.
- **Output**: Choose **Auto**, **Yes**, or **No** to decide whether to include the fixed page layout in the XML output. Choosing **Auto** exports the contents of the fixed layout.

Data Visualizers

There are several report controls that support a type of expression called Data Visualizers that allow you to create small graphs to make your data easy to understand. For example, you can red flag an overdue account using the Flags Icon Set as a background image in TextBox control.
There are several types of Data Visualizers available through a dialog linked to properties on the report controls.

Image Data Visualizers

These Data Visualizers are supported in the Image report control **Value** property, and also in the TextBox, CheckBox, Shape, and Container report controls' **BackgroundImage - Value** property. See the topics below to learn more.

> ⚠ **Caution**: In the following topics, the terms "argument" and "parameter" may seem interchangeable, but within an expression, an argument refers to the returned value of a parameter, while a parameter may be a variable.

Icon Set
: Learn about the included image strips from which you can select using arguments. These include traffic lights, arrows, flags, ratings, symbols, and more, plus you can create your own custom image strips.

Range Bar
: Learn how you can provide minimum, maximum, and length arguments to render a 96 by 96 dpi bar image in line with your text to show a quick visual representation of your data values.

Range Bar Progress
: Learn how you can use a second bar to show progress along with the data range.

Data Bar
: Learn about data bars, which are similar to range bars with a few different arguments.

Gradient
: Learn how to apply gradient style to the background of a control.

Hatch
: Learn how to apply hatch style to the background of a control.

Background Color Data Visualizer

These Data Visualizers are available in the TextBox and CheckBox report controls' **BackgroundColor** property. See the topics below to learn more.

Color Scale 2
: Learn about displaying TextBoxes with a range of background colors that are keyed to the value of the data.

Color Scale 3
: Learn about color scales with an additional middle color value.

Icon Set

The Icon Set data visualization allows you to use arguments to select an image from an image strip and display it either as a BackgroundImage, or as a Value. You can use the standard image strips included with ActiveReports, or create custom image strips.

The Icon Set Data Visualizer is supported in the Image report control **Value** property, and also in the TextBox, CheckBox, Shape, and Container report controls' **BackgroundImage - Value** property.

Standard Image Strips

Name	Image
Checkbox	
3TrafficLights	
Arrows	
Blank	
Flags	
GrayArrows	

Quarters	
Ratings	
RedToBlack	
Signs	
Symbols1	
Symbols2	
TrafficLights	

Note: When using icon sets, you must set the **Source** property to **Database**.

Parameters

- **Icon Set**. This designates the name of the icon set to use.
- **Icon 1 Value**. A Boolean expression that, if it evaluates to True, renders this icon from the strip.
- **Icon 2 Value**. A Boolean expression that, if it evaluates to True, renders this icon from the strip.
- **Icon 3 Value**. A Boolean expression that, if it evaluates to True, renders this icon from the strip.
- **Icon 4 Value**. A Boolean expression that, if it evaluates to True, renders this icon from the strip.
- **Icon 5 Value**. A Boolean expression that, if it evaluates to True, renders this icon from the strip.

You can use static values or any expression that evaluates to a Boolean value. For icon sets with fewer than five icons, set the unused values to False.

Syntax

```
=IconSet("Flags", False, True, False, False, False)
```

Usage

Following the Icon Set argument, there are five Boolean arguments. The first argument to evaluate to True displays the corresponding image. Use data expressions that evaluate to a Boolean value to replace the literal values in the code above.

Example

This expression displays the first symbol (the green flag) on each row in which the Difference exceeds 10, displays the second symbol (the yellow flag) on each row in which the quantity is greater than 0, and displays the third symbol (the red flag) on each row in which the quantity is equal to or below 0. Notice that we provide literal False values in the fourth and fifth arguments, which have no images in this strip.

Paste in the BackgroundImage Value property of a Textbox

```
=IconSet("Flags",Fields!Difference.Value > 10,Fields!Difference.Value >
0,Fields!Difference.Value <= 0,False,False)
```

Product ID	In Stock	Re Order Level	Difference	Icon Set
1000	5	7	-2	
1001	5	5	0	
1002	14	6	8	
1003	15	4	11	
1004	3	10	-7	
1005	5	8	-3	
1006	0	2	-2	
1007	6	7	-1	
1008	7	5	2	
1009	8	1	7	
1010	1	1	0	
1011	2	9	-7	
1012	4	1	3	
1013	0	8	-8	
1014	17	7	10	
1015	19	5	14	
1016	11	5	6	

In several of the included image strips, the last spots are empty. When using the Checkbox, 3TrafficLights, Flags, RedToBlack, Signs, Symbols1, Symbols2, or TrafficLights image strip, it generally makes sense to set the Boolean values for all of the unused icon spaces to False.

Custom Image Strips

The Blank image strip is included so that you can customize it. Drop down the section below for details.

Custom image strips

Custom image strips must conform to the following rules.

1. The format must be of a type that is handled by the .NET framework.
2. The size of the strip must be 120 x 24 pixels.
3. Each image must be 24 x 24 pixels in size.
4. There must be no more than five images in the strip.
5. If there are fewer than five images in the strip, there must be blank spaces in the image to fill in.

Types of Custom Images

Here is the syntax for various types of custom images, followed by examples of each.

External image syntax

```
=IconSet(location of image strip, condition#1, condition#2, condition#3, condition#4,
condition#5)
```

External image path example
`=IconSet("C:\Images\customstrip.bmp", 4 > 9, 5 > 9, 10 > 9, False, False)`

External image URL example
`=IconSet("http://mysite.com/images/customstrip.gif", 4 > 9, 5 > 9, 10 > 9, False, False)`

Image from an assembly resource syntax

```
=IconSet("res:[Assembly Name]/Resource name", condition#1, condition#2, condition#3,
condition#4, condition#5)
```

Assembly resource image example

`=IconSet("res:ReportAssembly,`
`Version=1.1.1.1./ReportAssembly.Resources.Images.CustomImage.png",`
`4 > 9, 5 > 9, 10 > 9, False, False)`

Embedded image syntax

```
=IconSet("embeddedImage:ImageName", condition#1, condition#2, condition#3, condition#4,
condition#5)
```

Embedded image example
`=IconSet("embeddedImage:Grades", Fields!Score.Value >=90, Fields!Score.Value >=80,` `Fields!Score.Value >=70, Fields!Score.Value >=60, True)`

Theme image syntax

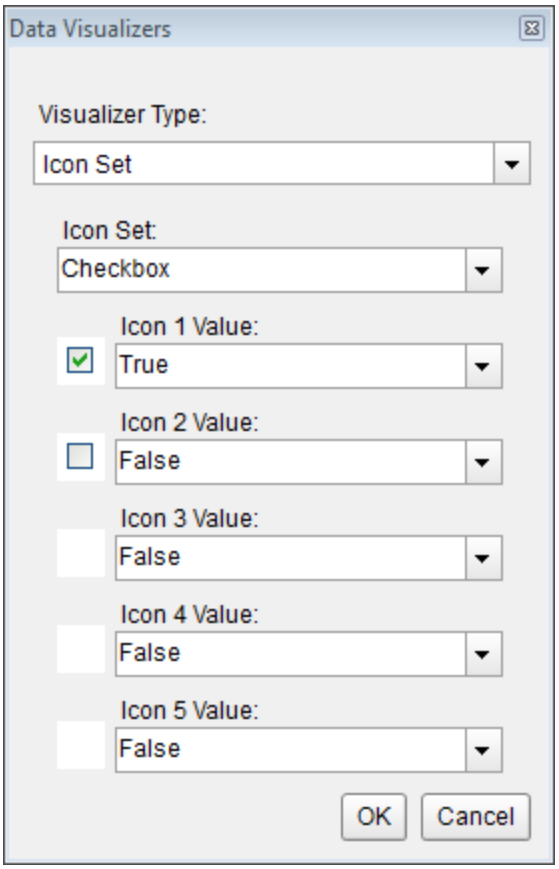
```
=IconSet("theme:ThemeImageName", condition#1, condition#2, condition#3, condition#4,
condition#5)
```

Theme image example

`=IconSet("theme:Grades", Fields!Score.Value >=90, Fields!Score.Value >=80,`
`Fields!Score.Value >=70,`
`Fields!Score.Value >=60, True)`

Data Visualizers Dialog

To open the dialog, drop down the **BackgroundImage** property of TextBox, CheckBox, Shape, and Container report controls, or the **Value** property of Image report control, and select **<Data Visualizer...>**. To build the data visualizer expression, select the appropriate values for each of the options in the dialog.



Range Bar

The Range Bar data visualization displays a 96 by 96 dpi bar image. The colored bar renders as half the height of the image, centered vertically. The amount of colored bar to render to the right of the Start argument (or to the left in the case of a negative value) is based on the Length argument. If the Length argument is zero, a diamond renders.

The Minimum and Maximum arguments determine the range of data. The area between the Length argument and the Maximum argument is transparent (or between the Length and the Minimum in the case of a negative value).

The Range Bar Data Visualizer is supported in the Image report control **Value** property, and also in the TextBox, CheckBox, Shape, and Container report controls' **BackgroundImage - Value** property.

Parameters

- **Minimum**. The lowest value in the range of data. This value corresponds to the leftmost edge of the image. If this argument is greater than the Start argument, Start becomes equal to Minimum. The data type is Single.
- **Maximum**. The highest value in the range of data. This value corresponds to the rightmost edge of the image. If this argument is less than the Start argument, Start becomes equal to Maximum. The data type is Single.
- **Color**. The HTML color string to use in rendering the Length in the bar image.
- **Start**. The point from which the Range Bar begins to be rendered. The data type is Single.
- **Length**. The width of the bar to render within the control. Setting this value to 0 renders a diamond shape instead

of a bar. The data type is Single.

You can use static values or aggregate functions (e.g. Min or Max) to set the parameters. For more information on these and other aggregate functions, see the Common Functions topic.

Syntax

```
=RangeBar(Minimum, Maximum, Color, Start, Length)
```

Usage

Use this data visualizer to render a bar in the color specified, the length of which changes depending on the number returned by the Length parameter, in the case of the simple example, GrossProfit. If your data contains only positive values, Start corresponds with Minimum at the left edge of the DataBar. The area between the Length and the Maximum is transparent.

Simple Example

Set the Length parameter to the value of a field in your dataset to display the field values visually.

Paste into a TextBox BackgroundImage property

```
=RangeBar(0,15000,"BlueViolet",0,Fields!GrossProfit.Value)
```

Store Name	Gross Profit	Range Bar
Store #1000	$12,602.57	
Store #1001	$10,348.09	
Store #1002	$13,939.84	
Store #1003	$12,201.39	
Store #1004	$11,383.74	
Store #1005	$10,979.59	
Store #1006	$12,709.01	

Example Using Negative Values

When your data contains negative as well as positive values, you can use an Immediate If expression for the Color parameter. In this example, if the Projected Stock value is negative, it renders in Crimson, while positive values render in BlueViolet. You can also see that negative values render to the left of Zero and positive values render to the right. A Length value of exactly zero renders as a diamond.

Paste into a TextBox BackgroundImage property

```
=RangeBar(-5,20,IIf((Fields!InStock.Value - 5) < 0, "Crimson",
"BlueViolet"),0,Fields!InStock.Value-5)
```

Title	In Stock	Projected Sales	Projected Stock	Range Bar
Snow White and the Seven Dwarfs	5	5	0	
Gone with the Wind	14	5	9	
Bambi	5	5	0	
One Hundred and One Dalmatians	7	5	2	
Mary Poppins	2	5	-3	
Doctor Zhivago	17	5	12	
The Jungle Book	7	5	2	
Butch Cassidy and the Sundance Kid	15	5	10	
Love Story	16	5	11	
The Godfather	11	5	6	
The Sting	8	5	3	
The Towering Inferno	6	5	1	
Blazing Saddles	7	5	2	
Jaws	11	5	6	
One Flew Over the Cuckoo's Nest	3	5	-2	

Default Behavior

The function returns **null** (i.e. no image is rendered) in the following cases:

1. The **Maximum** is less than or equal to the **Minimum**.
2. The expression is placed in a property which does not take an image.
3. The **Source** property of the image is not set to **Database**.

The Start value changes in the following cases:

1. If the **Start** value is less than the **Minimum** value, **Start** is the same as **Minimum**.
2. If the **Start** value is greater than the **Maximum** value, **Start** is the same as **Maximum**.

The Length value changes in the following cases:

1. If the **Start** value plus the **Length** value is less than the **Minimum** value, **Length** becomes **Minimum** minus **Start**.
2. If the **Start** value plus the **Length** value is greater than the **Maximum** value, **Length** becomes **Maximum** minus **Start**.

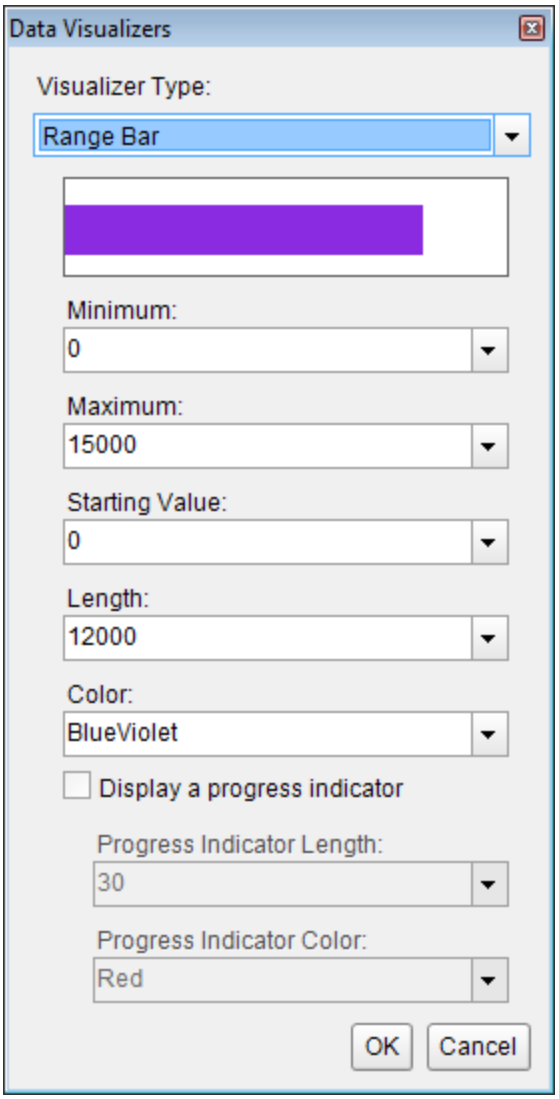
If the argument for any of the parameters cannot be converted to the required data type, the default value is used instead.

Parameter	Default Value
Minimum	0
Maximum	0
Color	Green

Start	0
Length	0

Dialog

When you select a TextBox, CheckBox, Shape, or a Container control on your report, in the Properties window or Properties dialog, you can drop down the **BackGroundImage Value** property and select **<Data Visualizer...>** to launch the dialog. The same is true if you select an Image control and drop down the **Value** property. To build the data visualizer expression, select the appropriate values for each of the options in the dialog.



Range Bar Progress

The Range Bar Progress data visualization displays a 96 by 96 dpi double bar image. The first colored bar renders as half the height of the image, centered vertically. The amount of colored bar to render to the right of the Start argument (or to the left in the case of a negative value) is based on the Length argument. If the Length argument is zero, a diamond renders.

The second colored bar renders using the ProgressColor as one fourth of the height of the image, centered vertically over the Length bar. The amount of colored bar to render to the right of the Start argument (or to the left in the case of a negative value) is based on

the Progress argument. If the Progress argument is zero, a smaller diamond renders.

The Minimum and Maximum arguments determine the range of data. The area between the Length and Progress arguments and the Maximum argument is transparent (or between the Length and Progress and the Minimum in the case of a negative value).

The Range Bar Progress Data Visualizer is supported in the Image report control **Value** property, and also in the TextBox, CheckBox, Shape, and Container report controls' **BackgroundImage - Value** property.

Parameters

- **Minimum**. The lowest value in the range of data. This value corresponds to the leftmost edge of the image. If this argument is greater than the Start argument, Start becomes equal to Minimum. The data type is Single.
- **Maximum**. The highest value in the range of data. This value corresponds to the rightmost edge of the image. If this argument is less than the Start argument, Start becomes equal to Maximum. The data type is Single.
- **Color**. The HTML color string to use in rendering the Length, the thicker bar in the bar image.
- **Start**. The point from which the Range Bar Progress begins to be rendered. The data type is Single.
- **Length**. The length of the thicker bar to render within the control. Setting this value to 0 renders a diamond shape instead of a bar. The data type is Single.
- **ProgressColor**. The HTML color string to use in rendering the Progress, the thinner bar in the bar image.
- **Progress**. The length of the thinner bar to render within the control. Setting this value to 0 renders a diamond shape instead of a bar. The data type is Single.

You can use static values or aggregate functions (e.g. Min or Max) to set the parameters. For more information on these and other aggregate functions, see the Common Functions topic.

Syntax

```
=RangeBarProgress(Minimum, Maximum, Color, Start, Length, ProgressColor, Progress)
```

Usage

Use this data visualizer to render a double bar in the colors specified, the length of which changes depending on the number returned by the Length parameter for the thick bar, in the case of the simple example, GrossSales. The thin bar length is based on the value returned by the Progress parameter, in this case, GrossProfit. If your data contains only positive values, Start corresponds with Minimum at the left edge of the Range Bar. The area between the Length or Progress and the Maximum is transparent.

Simple Example

Set the Length and Progress parameters to the values of fields in your dataset to display the field values visually.

Paste into a TextBox BackgroundImage property

```
=RangeBarProgress(0,30000,"BlueViolet",0,Fields!GrossSales.Value,"Gold",Fields!GrossProfit.Value)
```

Store Name	Gross Sales	Gross Profit	Range Bar
Store #1000	$22,797.30	$12,602.57	
Store #1001	$18,889.32	$10,348.09	
Store #1002	$25,625.24	$13,939.84	
Store #1003	$22,386.34	$12,201.39	
Store #1004	$19,893.03	$11,383.74	
Store #1005	$19,775.52	$10,979.59	
Store #1006	$22,400.15	$12,709.01	

Example Using Negative Values

When your data contains negative as well as positive values, you can use an Immediate If expression for the Color parameter. In the example below, if the Difference value is negative, it is rendered in red, while positive values are rendered in gold. You can also see that negative values are rendered to the left of Zero and positive values are rendered to the right. A Length value of exactly zero is rendered as a diamond. The thicker blue violet bar represents the InStock value.

Paste into a TextBox BackgroundImage property

```
=RangeBarProgress(-10,20,"BlueViolet",0,Fields!InStock.Value,IIf(Fields!Difference.Value < 0,
"Red", "Gold"),Fields!Difference.Value)
```

Product ID	In Stock	Re Order Level	Difference	Range Bar
1000	5	7	-2	
1001	5	5	0	
1002	14	6	8	
1003	15	4	11	
1004	3	10	-7	
1005	5	8	-3	
1006	0	2	-2	
1007	6	7	-1	
1008	7	5	2	
1009	8	1	7	
1010	1	1	0	
1011	2	9	-7	

Default Behavior

The function returns **null** (i.e. no image is rendered) in any of the following cases:

1. The **Maximum** is less than or equal to the **Minimum**.
2. The expression is placed in a property which does not take an image.
3. The **Source** property of the image is not set to **Database**.

The Start value changes in the following cases:

1. If the **Start** value is less than the **Minimum** value, **Start** is the same as **Minimum**.
2. If the **Start** value is greater than the **Maximum** value, **Start** is the same as **Maximum**.

The Length value changes in the following cases:

1. If the **Start** value plus the **Length** value is less than the **Minimum** value, **Length** becomes **Minimum** minus **Start**.
2. If the **Start** value plus the **Length** value is greater than the **Maximum** value, **Length** becomes **Maximum** minus **Start**.

The Progress value changes in the following cases:

1. If the **Start** value plus the **Progress** value is less than the **Minimum** value, **Progress** becomes **Minimum** minus **Start**.
2. If the **Start** value plus the **Progress** value is greater than the **Maximum** value, **Progress** becomes **Maximum** minus **Start**.

If the argument for any of the parameters cannot be converted to the required data type, the default value is used instead.

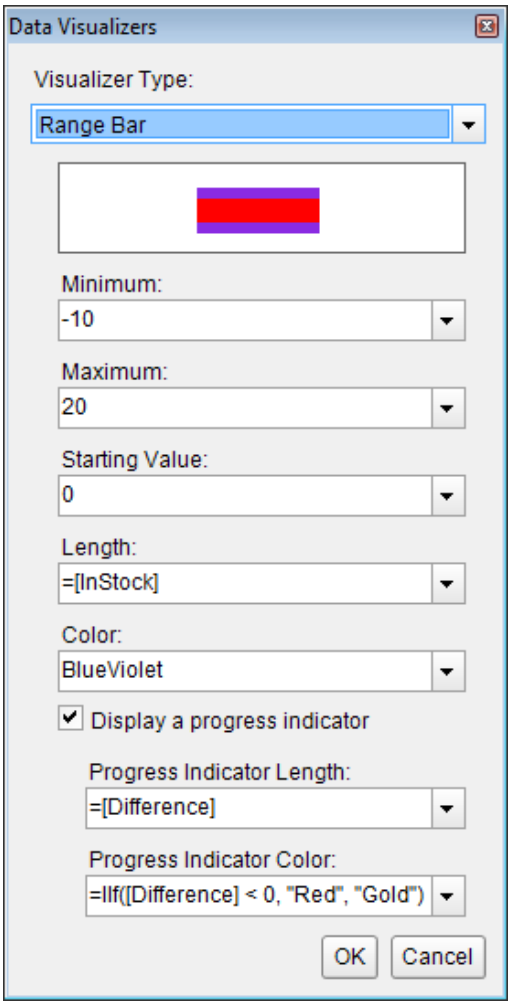
Parameter	Default Value

Minimum	0
Maximum	0
Color	Green
Start	0
Length	0
ProgressColor	Red
Progress	0

Dialog

When you select a TextBox, CheckBox, Shape, or a Container control on your report, in the Properties window or Properties dialog, you can drop down the **BackgroundImage Value** property and select **<Data Visualizer...>** to launch the dialog. To build the data visualizer expression, select the appropriate values for each of the options on the dialog.

For a Range Bar Progress expression, be sure to select the **Display a progress indicator** check box. This enables the progress options.



Data Bar

The Data Bar data visualization displays a 96 by 96 dpi bar image. The colored bar fills the Image top to bottom, while the Value argument determines the amount of colored bar to render to the right of the Zero argument (or to the left in the case of a negative value).

The Minimum and Maximum arguments determine the range of data. The area between the Value argument and the Maximum argument is transparent (or between the Value and the Minimum in the case of a negative value).

The Data Bar Data Visualizer is supported in the Image report control **Value** property, and also in the TextBox, CheckBox, Shape, and Container report controls' **BackgroundImage - Value** property.

Parameters

- **Value**. This is the field value in the report to be evaluated. The data type is Single.
- **Minimum**. The lowest value in the range of data against which the Value argument is compared. This value corresponds to the leftmost edge of the image. If this argument is greater than the Zero argument, Zero becomes equal to Minimum. The data type is Single.
- **Maximum**. The highest value in the range of data against which the Value argument is compared. This value corresponds to the rightmost edge of the image. If this argument is less than the Zero argument, Zero becomes equal to Maximum. The data type is Single.
- **Zero**. This value determines the zero point to the left of which negative data is rendered, and to the right of which positive data is rendered. The data type is Single.
- **Color**. The HTML color string to use in rendering the Value in the bar image.
- **Alternate Color**. The HTML color string to use when the Value is less than the Zero value (optional).

Select the **Use Alternate Color when Value is less than Zero Value** check box to enable the Alternate Color parameter. You can use static values or aggregate functions (e.g. Min or Max) to set parameters. For more information on these and other aggregate functions, see the Common Functions topic.

Syntax

```
=DataBar(Value, Minimum, Maximum, Zero, Color)
=DataBar(Value, Minimum, Maximum, Zero, Color, Alternate Color)
```

Usage

Use this data visualizer to render a bar in the color specified, the length of which changes depending on the number returned by the Value parameter, in the case of the simple example, InStock. If your data contains only positive values, Zero corresponds with Minimum at the left edge of the Data Bar. The area between the Value and the Maximum is transparent.

Simple Example

Set the Value parameter to the value of a field in your dataset to display the field values visually.

Paste into the BackgroundImage Value property of a TextBox
`=DataBar(Fields!InStock.Value,0,20,0,"BlueViolet")`

Product ID	In Stock	Data Bar
1000	5	
1001	5	
1002	14	
1003	15	
1004	3	
1005	5	
1006	0	
1007	6	
1008	7	
1009	8	
1010	1	
1011	2	
1012	4	

Example Using Negative Values

When your data contains negative as well as positive values, you can select the **Use Alternate Color when Value is less than Zero Value** check box, and then select an Alternate Color. In this example, if the Difference value is negative, it is rendered in Crimson, while positive values are rendered in BlueViolet. You can also see that negative values are rendered to the left of Zero and positive values are rendered to the right.

Paste in the BackgroundImage Value property of a TextBox

```
=DataBar(Fields!Difference.Value,-10,20,0,"BlueViolet","Crimson")
```

Product ID	In Stock	Re Order Level	Difference	Data Bar
1000	5	7	-2	
1001	5	5	0	
1002	14	6	8	
1003	15	4	11	
1004	3	10	-7	
1005	5	8	-3	
1006	0	2	-2	
1007	6	7	-1	
1008	7	5	2	
1009	8	1	7	
1010	1	1	0	
1011	2	9	-7	
1012	4	1	3	

Default Behavior

The function returns **null** (i.e. no image is rendered) in any of the following cases:

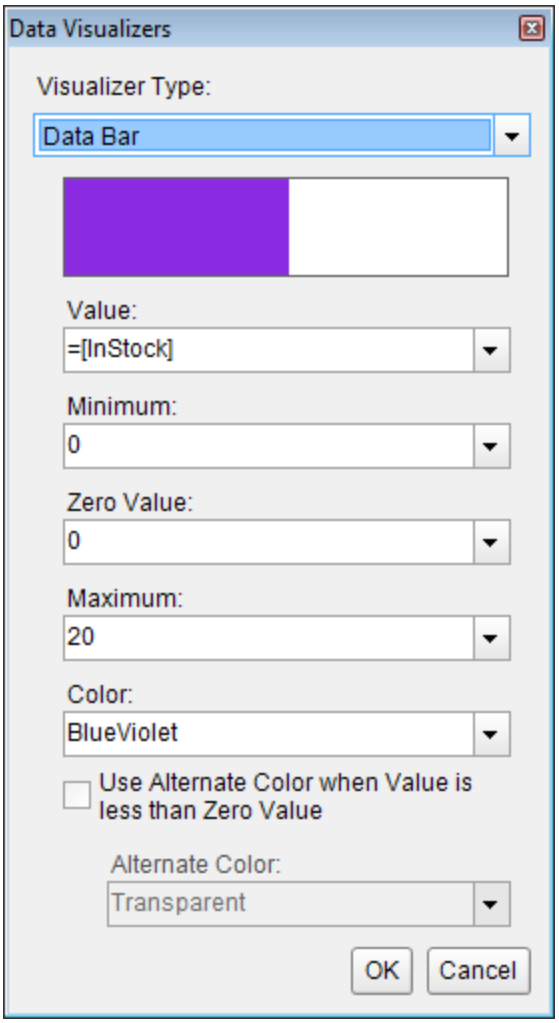
1. The **Maximum** is less than or equal to the **Minimum**.
2. The expression is placed in a property which does not take an image.
3. The **Source** property of the image is not set to **Database**.

If the argument for any of the parameters cannot be converted to the required data type, the default value is used instead.

Parameter	Default Value
Value	0
Minimum	0
Maximum	0
Zero	0
Color	Green
Alternate Color	*null*

Dialog

When you select a TextBox, CheckBox, Shape, or a Container control on your report, in the Properties window or Properties dialog, you can drop down the **BackGroundImage Value** property and select **<Data Visualizer...>** to launch the dialog. The same is true if you select an Image control and drop down the **Value** property. To build the data visualizer expression, select the appropriate values for each of the options in the dialog.



Gradient

The Gradient data visualization displays a gradient of color that transitions from one color to another.

Parameters

- **Gradient type**. The available gradient types are Horizontal, Vertical, DiagonalUp, DiagonalDown, FromCenter, and FromCorner.
- **Color1**. The start (primary) color of the gradient.
- **Color2**. The end (secondary) color of the gradient.

Syntax

```
=Gradient(Gradient type, Color1, Color2)
```

Usage

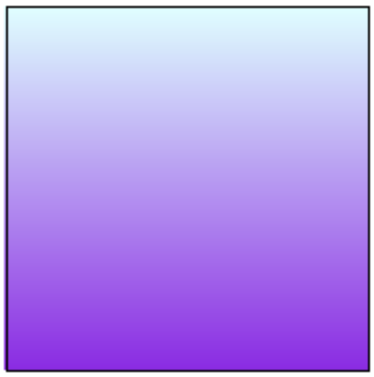
Using Data Visualizers, the gradient can be applied in TextBox, Shape, CheckBox, and Container

controls from **BackgroundImage - Value** property and in Image control from **Value** property. The following example shows how Gradient background style can be simply applied to a control.

Example

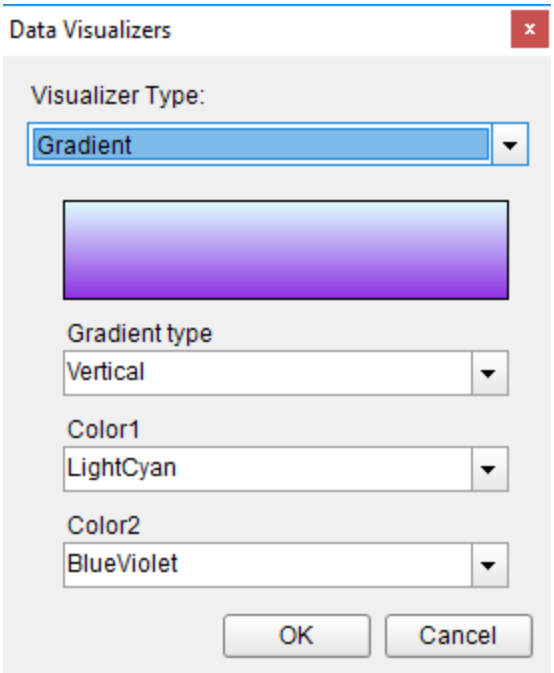
Set the following expression in the BackgroundImage.Value property of the Shape control.

Paste into the BackgroundImage Value property of a Shape
`=Gradient("Vertical","LightCyan","BlueVoilet")`



Dialog

When you select a Shape control on your report, in the Properties window, drop down the BackGroundImage - Value property and select <Data Visualizer...> to launch the dialog. The same is true if you select an Image control and drop down the Value property. To build the data visualizer expression, select the appropriate values for each of the options in the dialog.



Note: The Gradient style set using Data Visualizers is not displayed at design time.

Hatch

The Hatch data visualization displays the geometric hatch pattern.

Parameters

- **Hatch style**. The available hatch styles are Horizontal, Vertical, ForwardDiagonal, BackwardDiagonal, LargeGrid, and many more.
- **Color1**. The color of pattern or foreground color.
- **Color2**. The background color.

Syntax

```
=Hatch(Hatch style,Color1,Color2)
```

Usage

Using Data Visualizers, the hatch can be applied in TextBox, Shape, CheckBox, and Container controls from **BackgroundImage - Value** property and in Image control from **Value** property. The following example shows how Hatch background style can be simply applied to a control.

Example

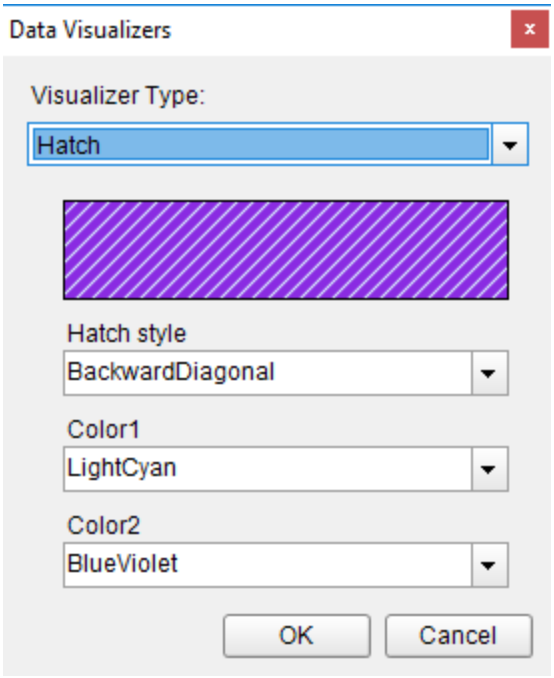
Set the following expression in the Shape control's BackgroundImage - Value property.

Paste into BackgroundImage property
`=Hatch("BackwardDiagonal","LightCyan","BlueVoilet")`

Dialog

When you select a Shape control on your report, in the Properties window, drop down the BackGroundImage - Value property and select <Data Visualizer...> to launch the dialog. The same is true if you select an Image control and drop down the Value property. To build the data visualizer expression, select the appropriate values for each of the options in

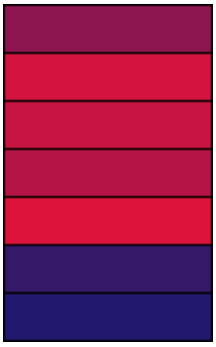
the dialog.



Note: The Hatch style set using Data Visualizers is not displayed at design time.

Color Scale 2

The ColorScale2 data visualization displays a background color in a range of colors to indicate minimum and maximum values, and all shades in between. The Color Scale Data Visualizer is available for TextBox and CheckBox controls.



Parameters

- **Value**. This is the field value in the report to evaluate. The data type is Single.
- **Minimum**. If the Value evaluates to this number, the StartColor renders.
- **Maximum**. If the Value evaluates to this number, the EndColor renders.
- **StartColor**. The HTML color string to use if the Value evaluates to the Minimum value.
- **EndColor**. The HTML color string to use if the Value evaluates to the Maximum value.

You can use static values or aggregate functions (e.g. Min or Max) to set the Minimum and Maximum parameters. For

more information on these and other aggregate functions, see the Common Functions topic.

Syntax

```
=ColorScale2(Value, Minimum, Maximum, StartColor, EndColor)
```

Usage

Use an expression with this syntax in the **BackgroundColor** property of a Textbox or a CheckBox control. This causes the background color to change depending on the value of the field you specified in the **Value** parameter, in the case of the example, InStock. Any values falling between the **Minimum** value and the **Maximum** render with a color between the **StartColor** and **EndColor**.

Example

Set the Value parameter to the value of a field in your dataset to display the field values visually.

Paste into the BackgroundColor property of a TextBox
`=ColorScale2(Fields!InStock.Value,0,20,"Crimson","MidnightBlue")`

Product ID	In Stock	Color Scale 2
1000	5	
1001	5	
1002	14	
1003	15	
1004	3	
1005	5	
1006	0	
1007	6	
1008	7	
1009	8	
1010	1	
1011	2	
1012	4	
1013	0	
1014	17	
1015	19	

Default Behavior

The function returns **Transparent** in any of the following cases:

1. The **Value** is out of range (i.e. does not fall between the **Minimum** and **Maximum** values).
2. The **Maximum** is less than the **Minimum**.

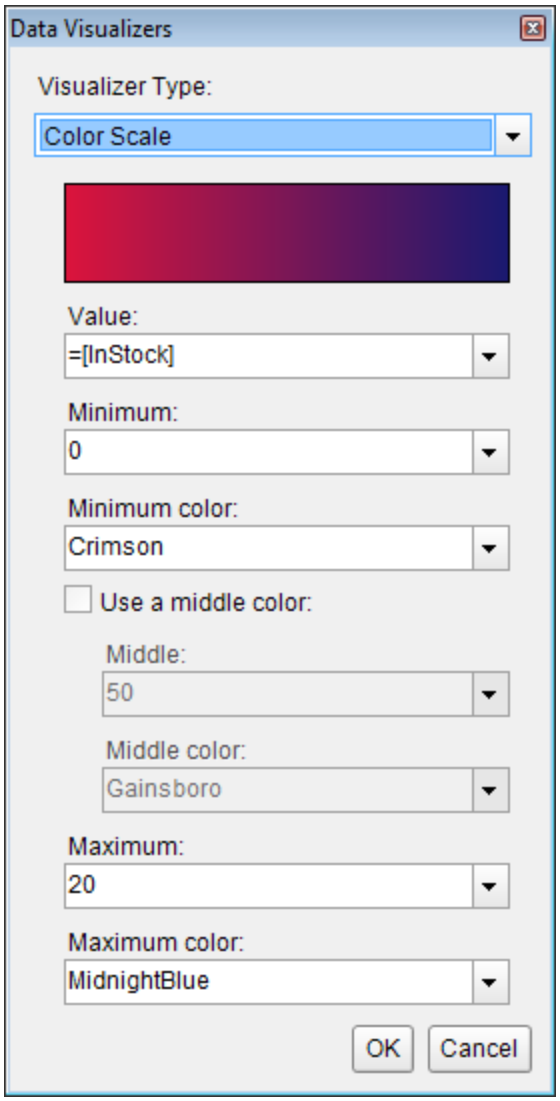
If the argument for any of the parameters cannot be converted to the required data type, the default value is used instead.

Parameter	Default Value
Value	0
Minimum	0
Maximum	0
StartColor	Silver
EndColor	WhiteSmoke

Dialog

When you select a TextBox or a CheckBox control on your report, in the Properties window or Properties dialog, you can drop down the **BackGroundColor** property and select **<Data Visualizer...>** to launch the dialog. To build the data visualizer expression, select the appropriate values for each of the options in the dialog.

> **Note**: If you select the **Use a middle color** check box, the expression used in the BackgroundColor property changes to ColorScale3. For more information, see ColorScale3.



Color Scale 3

The ColorScale3 data visualization displays a background color in a range of colors to indicate minimum, middle, and maximum values, and all shades in between. The Color Scale Data Visualizer is available for TextBox and CheckBox controls.

Parameters

- **Value**. This is the field value in the report to be evaluated. The data type is Single.
- **Minimum**. If the Value evaluates to this number, the StartColor is rendered.
- **Middle**. If the Value evaluates to this number, the MiddleColor is rendered.
- **Maximum**. If the Value evaluates to this number, the EndColor is rendered.
- **StartColor**. The HTML color string to use if the Value evaluates to the Minimum value.
- **MiddleColor**. The HTML color string to use if the Value evaluates to the Middlevalue.
- **EndColor**. The HTML color string to use if the Value evaluates to the Maximum value.

You can use static values or aggregate functions (e.g. Min, Avg, or Max) to set the Minimum, Middle, and Maximum parameters. For more information on these and other aggregate functions, see the Common Functions topic.

Syntax

```
=ColorScale3(Value, Minimum, Middle, Maximum, StartColor, MiddleColor, EndColor)
```

Usage

Use an expression with this syntax in the **BackgroundColor** property of a Textbox or a CheckBox control. This causes the background color to change depending on the value of the field you specified in the **Value** parameter, in the case of the example, InStock. Any values falling between the **Minimum** value and the **Middle** value render with a gradient scale color between the **StartColor** and **MiddleColor**. The closer the value is to the **Minimum**, the closer to Crimson the color renders. In the same way, values falling between the **Middle** and **Maximum** render with a color between the **MiddleColor** and **EndColor**, in this case, varying shades of yellow-green.

Example

Set the Value parameter to the value of a field in your dataset to display the field values visually.

Paste into the BackgroundColor property of a TextBox

```
=ColorScale3(Fields!InStock.Value,0,10,20,"Crimson","Yellow","MidnightBlue")
```

Product ID	In Stock	Color Scale 3
1000	5	
1001	5	
1002	14	
1003	15	
1004	3	
1005	5	
1006	0	
1007	6	
1008	7	
1009	8	
1010	1	
1011	2	
1012	4	
1013	0	
1014	17	
1015	19	
1016	11	

Default Behavior

The function returns **Transparent** in any of the following cases:

1. The **Value** is out of range (i.e. does not fall between the **Minimum** and **Maximum** values).
2. The **Maximum** is less than the **Minimum**.
3. The **Middle** is not between the **Minimum** and the **Maximum**.

If the argument for any of the parameters cannot be converted to the required data type, the default value is used instead.

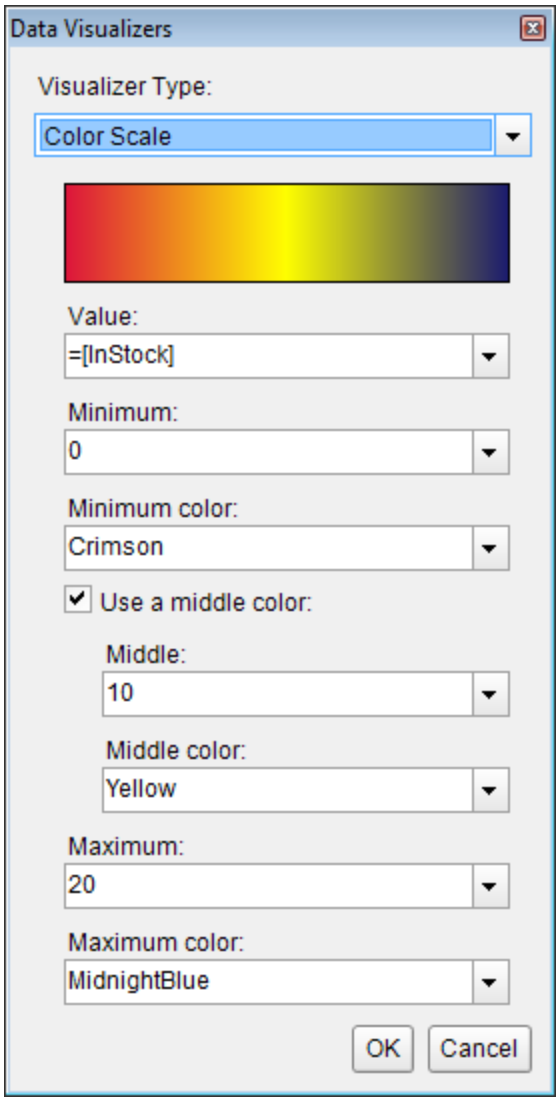
Parameter	Default Value
Value	0
Minimum	0
Middle	0
Maximum	0
StartColor	Silver
MiddleColor	Gainsboro

EndColor	WhiteSmoke

Dialog

When you select a TextBox or a CheckBox control on your report, in the Properties window or Properties dialog, you can drop down the **BackGroundColor** property and select **<Data Visualizer...>** to launch the dialog. To build the data visualizer expression, select the appropriate values for each of the options in the dialog.

> **Note**: If you clear the **Use a middle color** check box, the expression used in the BackgroundColor property changes to ColorScale2. For more information, see ColorScale2.



Custom Resource Locator

Page reports and RDL reports can resolve resources from your file system using file paths, but sometimes resources are preserved in very specific sources, such as a database. With RDL report, you can create a custom resource locator to read any resources that might be required by your reports from any type of location. You can use it for resources such

as images and theme files, or for reports to use in drillthrough links, subreports, or master reports.

API

You can implement a custom resource locator by deriving from the **GrapeCity.ActiveReports.ResourceLocator** and overriding the **GetResource** method.

The GetResource method returns **ParentUri** and **Value** properties. The Value property contains the located resource as a memory stream. The ParentUri property contains the string URI of the parent of the resource within the resource hierarchy.

Here is the GetResource method used in the sample.

C# MyPicturesLocator.cs code showing usage of the GetResource Method from the Sample

C# code. Paste inside a class.

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Globalization;
using System.IO;
using System.Runtime.InteropServices;
using System.Text;
using GrapeCity.ActiveReports.Samples.CustomResourceLocator.Properties;
namespace GrapeCity.ActiveReports.Samples.CustomResourceLocator
{
    /// Implementation of ResourceLocator which looks for resources in the My Pictures
folder.
    internal sealed class MyPicturesLocator : ResourceLocator
    {
        private const string UriSchemeMyImages = "MyPictures:";
    '<summary>
    'Look for the resources in My Pictures folder.
    '</summary>
    '<param name="resourceInfo">The information about the resource to be obtained.
</param>
    '<returns>The resource, null if it was not found. </returns>
        public override Resource GetResource(ResourceInfo resourceInfo)
        {

            Resource resource;
            string name = resourceInfo.Name;
            if (name == null || name.Length == 0)
            {
                throw new ArgumentException(Resources.ResourceNameIsNull, "name");
            }
            Uri uri = new Uri(name);
            if (uri.GetLeftPart(UriPartial.Scheme).StartsWith(UriSchemeMyImages, true,
CultureInfo.InvariantCulture))
            {
```

```
                Stream stream = GetPictureFromSpecialFolder(uri);
                if (stream == null)
                {
                    stream = new MemoryStream();
                    Resources.NoImage.Save(stream, Resources.NoImage.RawFormat);
                }
                resource = new Resource(stream, uri);
            }
            else
            {
                throw new
InvalidOperationException(Resources.ResourceSchemeIsNotSupported);
            }
            return resource;
        }
            /// <summary>
        /// Returns the specified image from My Pictures folder.
        /// </summary>
        /// <param name="path">The uri of the image located in My Pictures code, i.e.
MyImages:logo.gif.</param>
        /// <returns>The stream contains the image data or null if the picture can't be
found or handled.</returns>
        private static Stream GetPictureFromSpecialFolder(Uri path)
        {
            int startPathPos = UriSchemeMyImages.Length;
            if (startPathPos >= path.ToString().Length)
            {
                return null;
            }
            string pictureName = path.ToString().Substring(startPathPos);
            string myPicturesPath =
Environment.GetFolderPath(Environment.SpecialFolder.MyPictures);
            if (!myPicturesPath.EndsWith("\\")) myPicturesPath += "\\";
             string picturePath = Path.Combine(myPicturesPath, pictureName);
            if (!File.Exists(picturePath)) return null;
            MemoryStream stream = new MemoryStream();
            try
            {
                Image picture = Image.FromFile(picturePath);
                picture.Save(stream, picture.RawFormat);
                stream.Position = 0;
            }
            catch(OutOfMemoryException)
                        /// The file is not a valid image, or GDI+ doesn't support the
image type.
            {
                return null;
            }
```

```
            catch(ExternalException)
                        /// The image can't be saved.
            {
                return null;
            }
            return stream;
        }
    }
}
```

Visual Basic MyPicturesLocator.vb code showing usage of the GetResource Method from the Sample

Visual Basic code. Paste inside a class.

```
Imports System.Globalization
Imports System.IO
Imports System.Runtime.InteropServices

Public Class MyPicturesLocator
    Inherits ResourceLocator

    Const UriSchemeMyImages As String = "MyPictures:"

    '<summary>
    'Look for the resources in My Pictures folder.
    '</summary>
    '<param name="resourceInfo">The information about the resource to be obtained.
</param>
    '<returns>The resource, null if it was not found. </returns>

    Public Overrides Function GetResource(ByVal resourceInfo As ResourceInfo) As
Resource
        Dim resource As Resource
        Dim name As String = resourceInfo.Name

        If (String.IsNullOrEmpty(name)) Then

            Throw New ArgumentException(My.Resources.ResourceNameIsNull, "name")

        End If

        Dim uri As New Uri(name)
        If (Uri.GetLeftPart(UriPartial.Scheme).StartsWith(UriSchemeMyImages, True,
CultureInfo.InvariantCulture)) Then

            Dim stream As Stream = GetPictureFromSpecialFolder(uri)
            If (stream Is Nothing) Then
                stream = New MemoryStream()
```

```
                My.Resources.NoImage.Save(stream, My.Resources.NoImage.RawFormat)
            End If
            resource = New Resource(stream, uri)

        Else
            Throw New
InvalidOperationException(My.Resources.ResourceSchemeIsNotSupported)
        End If
        Return resource
    End Function

    Function GetPictureFromSpecialFolder(ByVal path As Uri) As Stream
        Dim startPathPos As Integer = UriSchemeMyImages.Length

        If (startPathPos >= path.ToString().Length) Then
            Return Nothing
        End If

        Dim pictureName As String = path.ToString().Substring(startPathPos)
        Dim myPicturesPath As String =
Environment.GetFolderPath(Environment.SpecialFolder.MyPictures)

        If (Not myPicturesPath.EndsWith("\\")) Then
            myPicturesPath += "\\"
        End If

        Dim picturePath As String = System.IO.Path.Combine(myPicturesPath, pictureName)

        If (Not File.Exists(picturePath)) Then
            Return Nothing
        End If

        Dim stream As New MemoryStream()

        Try
            Dim picture As Image = Image.FromFile(picturePath)
            picture.Save(stream, picture.RawFormat)
            stream.Position = 0
        Catch ex As OutOfMemoryException
                ' The file is not a valid image, or GDI+ doesn't support the image type.
            Return Nothing
        Catch ex As ExternalException
                ' The image can't be saved.
            Return Nothing
        End Try

        Return stream
    End Function
```

```
End Class
```

Sample

The product sample is located here:

`..\Samples14\Advanced\PageAndRDL\CustomResourceLocator\C#` or VB.NET

contains a custom resource locator that looks for files in the current user's My Pictures folder. It does this by looking for special MyPictures protocol.

Section Report Concepts

There are a number of concepts that only apply to section reports.

In this section

Section Report Toolbox
: This section provides information on each of the report controls available in the ActiveReports 14 **Section Report** group of the Visual Studio toolbox.

Section Report Structure
: Learn about the report structure in a section layout.

Section Report Events
: Learn about events that you can use to customize section reports.

Designing Code-based Section Reports in .NET Core
: Learn about the workaround to design Code-Based Section Reports in projects targeting .NET Core.

Scripting in Section Reports
: Learn how to use scripts in a section layout.

Report Settings Dialog
: See the various options provided in Report Settings dialog.

Date, Time, and Number Formatting
: Learn how you can customize formatting with .NET strings.

Optimizing Section Reports
: Learn about ways to optimize section reports to reduce memory consumption and increase speed.

CacheToDisk and Resource Storage
: Learn about IsolatedStorage and other considerations when you use CacheToDisk to reduce memory consumption.

Section Report Toolbox

When a Section report has focus in Visual Studio, the ActiveReports 14 **Section Report** toolbox group offers a number of report controls that you can use when creating a section report. You can drag these from the toolbox and drop them onto your section reports. These tools are different than those in the Toolbox.

> **Note:** Take care in naming report controls, as they are displayed to end users in the advanced search feature of the Viewer.

In this section

Label
: The label is used to display descriptive text for a control and helps the user to describe the data displayed in a report.

TextBox (Section Report)
: The text box is a basic reporting control that allows direct display and editing of unformatted text.

CheckBox (Page Report)
: The checkbox gives the user an option of yes or no and true or false.

RichTextBox
: The rich text box control allows the user to enter rich text in the form of formatted text, tables, hyperlinks, images, etc.

Shape
: The shape is a user interface element that allows to draw shapes on the screen.

Picture
: This control displays images files on the screen and also performs functions like resizing and cropping of images being used.

Line (Section Report)
: The line visually draws boundaries or highlights specific areas of a report.

Page Break
: The PageBreak control is used when you need to stop printing a report inside the selected section and resume it on a new page.

Barcode (Section Report)
: The BarCode control allows you to choose from several barcode styles available, and bind them to a data source.

SubReport (Section Report)
: Use the subreport control as a placeholder for data from a separate report. Use code to connect the separate report to the subreport control.

Ole Object
: You can add an OLE object, bound to a database or unbound, directly to your report.

> **Note**: The OleObject control is not displayed in the toolbox by default, because it is obsolete, and is only available for backward compatibility.

Chart
: You can use the ChartControl for a graphical presentation of data in a report. There are numerous chart types that you can use to easily design and render data.

Report Info
: The ReportInfo control allows you to quickly display page numbers, page counts and report dates.

Cross Section Controls
: The CrossSectionLine and CrossSectionBox controls provide visual boundaries and highlight specific areas of your report that span multiple report sections. This CrossSectionLine control is a vertical line that starts in the header section and spans the intervening sections until it ends in the footer. (For a horizontal or diagonal line, use the **Line** control.) The CrossSectionBox control starts in the header section and spans any intervening sections to end in the related footer section.

Label

The Label control for Section reports is very similar to the standard Visual Studio Label control. Since, by inheriting from the ARControl object, it can bind to data with the DataField property, and since you can enter static text in the TextBox control, the main difference between the two controls is the **Angle** property of the Label control, and the following

properties of the TextBox control: CanGrow, CanShrink, CountNullValues, Culture, DistinctField, OutputFormat, SummaryFunc, SummaryGroup, SummaryRunning, and SummaryType.

This control may become obsolete if the Angle property is added to the TextBox, and then will be kept only for compatibility with previous versions.

Important Properties

Property	Description
Angle	Gets or sets the angle (slope) of the text within the control area. Set the **Angle** property to 900 to display text vertically.
CharacterSpacing	Gets or sets the space between characters in points.
DataField	Gets or sets the field name from the data source to bind to the control.
HyperLink	Gets or sets a URL to which the viewer navigates when the user clicks the label at run time. The URL becomes an anchor tag or a hyperlink in HTML and PDF exports.
LineSpacing	Gets or sets the space between lines in points.
MinCondenseRate	Specifies the minimal rate of the text horizontal shrinking in percentages. Should be between 10 and 100.
MultiLine	Gets or sets a value indicating whether to allow text to break to multiple lines within the control.
ShrinkToFit	Gets or sets a value indicating whether to decrease the font size so that all of the text shows within the boundaries of the control.
Style	Gets or sets a style string for the label.
Text	Gets or sets the text to show on the report.
TextJustify	Specifies how to distribute text when the Alignment property is set to Justify. With any other Alignment setting, this property is ignored.
VerticalAlignment	Gets or sets the vertical position of the label's text within the bounds of the control.
VerticalText	Indicates whether to render the label's text vertically.
WrapMode	Indicates whether a multi-line label control wraps words or characters to the beginning of the next line when necessary.

You can double-click in the Label control to enter edit mode and enter text directly in the control, or you can enter text in the Properties window or in code through the **Text** property.

You can format text in the Label control in edit mode using the ActiveReports toolbar, or you can modify properties in the Properties window. Formats apply to all of the text in the control. Text formatting changes in the Properties window immediately appear in the control, and changes made in the toolbar are immediately reflected in the Properties window.

> **Note**: In edit mode for a Label with the Alignment property set to Justify, the Alignment value temporarily changes to the default value, Left. Once you leave edit mode, it automatically changes back to Justify.

Keyboard Shortcuts

In edit mode, you can use the following keyboard shortcuts.

Key Combination	Action
Enter	New line.
Alt + Enter	Saves modifications and exits edit mode.
Esc	Cancels modifications and exits edit mode.

In the End User Designer, you can disable this feature in the **EditModeEntering ('EditModeEntering Event' in the on-line documentation)** and **EditModeExit ('EditModeExit Event' in the on-line documentation)** events.

Label Dialog

With the control selected on the report, in the Commands section at the bottom of the Properties window, you can click the **Property dialog** command to open the dialog.

General

Name: Enter a name for the label that is unique within the report. This name is displayed in the Document Outline and in XML exports. You can only use underscore (_) as a special character in the Name field. Other special characters such as period (.), space ( ), forward slash (/), back slash (\), exclamation (!), and hyphen (-) are not supported.

Tag: Enter a string that you want to persist with the control. If you access this property in code, it is an object, but in the Properties window or Property dialog, it is a string.

Visible: Clear this check box to hide the control.

DataField: Select a field from the data source to bind to the control.

Text: Enter static text to show in the label.

HyperLink: Enter a URL to use in the Viewer HyperLink event. The URL automatically converts to an anchor tag or hyperlink in PDF and HTML exports.

Appearance

Background Color: Select a color to use for the background of the label.

Angle: Use the slider to set the degree of slope for the text within the control area.

Font

Name: Select a font family name or a theme font.

Size: Choose the size in points for the font.

Style: Choose **Normal** or **Italic**.

Weight: Choose from **Normal** or **Bold**.

Color: Choose a color to use for the text.

Decoration: Select check boxes for **Underline** and **Strikeout**.

GDI Charset: Enter a value to indicate the GDI character set to use. For a list of valid values, see MSDN Font.GDICharSet Property.

GDI Vertical: Select this checkbox to indicate that the font is derived from a GDI vertical font.

Format

Line spacing: Enter a value in points to use for the amount of space between lines.

Character spacing: Enter a value in points to use for the amount of space between characters.

Multiline: Select this check box to allow text to render on multiple lines within the control.

Minimal rate of text horizontal shrinking (in %): Specify the percentage to which the text should be shrunk horizontally.

Text direction

RightToLeft: Select this check box to reverse the text direction.

Vertical text: Select this check box for top to bottom text.

Alignment

Vertical alignment: Choose **Top**, **Middle**, or **Bottom**.

Horizontal alignment: Choose **Left**, **Center**, **Right**, or **Justify**.

Justify method: Choose **Auto**, **Distribute**, or **DistributeAllLines**.

> **Note:** You must select **Justify** in the **Horizontal alignment** property to enable the **Justification method** property options.

Wrap mode: Choose **NoWrap**, **WordWrap**, or **CharWrap** to determine whether and how text breaks to the next line.

Padding

Enter values in points to set the amount of space to leave around the label.

- **Top**
- **Left**
- **Right**
- **Bottom**

TextBox (Section Report)

The TextBox control is the basis of reporting as it is used to display text in section reports. You can bind it to data or set it at run time. It is the same control that forms when you drag a field onto a report from the Report Explorer.

Important Properties

Property	Description
CharacterSpacing	Gets or sets a character spacing in points.
LineSpacing	Gets or sets a line spacing in points.
OutputFormat	Gets or sets the mask string used to format the Value property before placing it in the Text property.
Style	Gets or sets a style string for the textbox.
TextJustify	Specifies text justification with TextAlign set to Justify.

VerticalAlignment	Gets or sets the position of the textbox's text vertically within the bounds of the control.
VerticalText	Gets or sets whether to render text according to vertical layout rules.
CanGrow	Determines whether ActiveReports should increase the height of the control based on its content.
CanShrink	Determines whether ActiveReports should decrease the height of the control based on its value.
MinCondenseRate	Specifies the minimal rate of the text horizontal shrinking in percentages. Should be between 10 and 100.
MultiLine	Gets or sets a value indicating whether this is a multi-line textbox control.
ShrinkToFit	Determines whether ActiveReports decreases the font size when text values exceed available space.
WrapMode	Indicates whether a multi-line textbox control automatically wraps words or characters to the beginning of the next line when necessary.
CountNullValues	Boolean which determines whether DBNull values should be included as zeroes in summary fields.
Culture	Gets or sets CultureInfo used for value output formatting.
DataField	Gets or sets the field name from the data source to bind to the control.
HyperLink	Gets or sets the hyperlink for the text control.
Text	Gets or sets the formatted text value to be rendered in the control.
DistinctField	Gets or sets the name of the data field used in a distinct summary function.
SummaryFunc	Gets or sets the summary function type used to process the DataField Values.
SummaryGroup	Gets or sets the name of the group header section that is used to reset the summary value when calculating subtotals.
SummaryRunning	Gets or sets a value that determines whether that data field summary value will be accumulated or reset for each level (detail, group or page).
SummaryType	Gets or sets a value that determines the summary type to be performed.

You can double-click in the TextBox control to enter edit mode and enter text directly in the control, or you can enter text in the Properties window or in code through the **Text** property.

You can format text in the TextBox control in edit mode using the ActiveReports toolbar, or you can modify properties in the Properties window. Formats apply to all of the text in the control. Text formatting changes in the Properties window immediately appear in the control, and changes made in the toolbar are immediately reflected in the Properties window.

> **Note**: In edit mode for a TextBox with the Alignment property set to Justify, the Alignment value temporarily changes to the default value, Left. Once you leave edit mode, it automatically changes back to Justify.

Keyboard Shortcuts

In edit mode, you can use the following keyboard shortcuts.

Key Combination	Action

Enter	New line.
Alt + Enter	Saves modifications and exits edit mode.
Esc	Cancels modifications and exits edit mode.

In the End User Designer, you can disable this feature in the **EditModeEntering ('EditModeEntering Event' in the on-line documentation)** and **EditModeExit ('EditModeExit Event' in the on-line documentation)** events.

TextBox Dialog

With the control selected on the report, in the Commands section at the bottom of the Properties window, you can click the **Property dialog** command to open the dialog.

General

Name: Enter a name for the textbox that is unique within the report. This name is displayed in the Document Outline and in XML exports. You can only use underscore (_) as a special character in the Name field. Other special characters such as period (.), space ( ), forward slash (/), back slash (\), exclamation (!), and hyphen (-) are not supported.

Tag: Enter a string that you want to persist with the control. If you access this property in code, it is an object, but in the Properties window or Property dialog, it is a string.

Visible: Clear this check box to hide the control.

DataField: Select a field from the data source to bind to the control.

Text: Enter static text to show in the textbox. If you specify a DataField value, this property is ignored.

HyperLink: Enter a URL to use in the Viewer HyperLink event. The URL automatically converts to an anchor tag or hyperlink in PDF and HTML exports.

Appearance

Background Color: Select a color to use for the background of the textbox.

Font

Name: Select a font family name or a theme font.

Size: Choose the size in points for the font.

Style: Choose **Normal** or **Italic**.

Weight: Choose from **Normal** or **Bold**.

Color: Choose a color to use for the text.

Decoration: Select check boxes for **Underline** and **Strikeout**.

GDI Charset: Enter a value to indicate the GDI character set to use. For a list of valid values, see MSDN Font.GDICharSet Property.

GDI Vertical: Select this checkbox to indicate that the font is derived from a GDI vertical font.

Format

Line spacing: Enter a value in points to use for the amount of space between lines.

Character spacing: Enter a value in points to use for the amount of space between characters.

Multiline: Select this check box to allow text to render on multiple lines within the control.

Minimal rate of text horizontal shrinking (in %): Specify the percentage to which the text should be shrunk horizontally.

Textbox height

Can increase to accommodate contents: Select this check box to set CanGrow to True.

Can decrease to accommodate contents: Select this check box to set CanShrink to True.

Can shrink text to fit fixed size control: Select this check box to set ShrinkToFit to True.

Text direction

RightToLeft: Select this check box to reverse the text direction.

Vertical text: Select this check box for top to bottom text.

Alignment

Vertical alignment: Choose **Top**, **Middle**, or **Bottom**.

Horizontal alignment: Choose **Left**, **Center**, **Right**, or **Justify**.

Justify method: Choose **Auto**, **Distribute**, or **DistributeAllLines**.

Wrap mode: Choose NoWrap, WordWrap, or CharWrap to select whether to wrap words or characters to the next line.

> **Note:** You must select **Justify** in the **Horizontal alignment** property to enable the **Justification method** property options.

Wrap mode: Choose **NoWrap**, **WordWrap**, or **CharWrap** to determine whether and how text breaks to the next line.

Padding

Enter values in points to set the amount of space to leave around the label.

- **Top**
- **Left**
- **Right**
- **Bottom**

Summary

SummaryFunc: Select a type of summary function to use if you set the SummaryRunning and SummaryType properties to a value other than None. For descriptions of the available functions, see the **SummaryFunc Enumeration (on-line documentation)**.

SummaryGroup: Select a section group that you have added to the report. If you also set the SummaryRunning property to Group, the textbox summarizes only the values for that group.

SummaryRunning: Select **None**, **Group**, or **All**. If None, the textbox shows the value for each record. If Group, the textbox summarizes the value for the selected SummaryGroup. If All, the textbox summarizes the value for the entire report.

SummaryType: Select the type of summary to use. For descriptions of the available types, see the **SummaryType**

Enumeration (on-line documentation).

Distinct Field: Select a field to use with one of the SummaryFunc distinct enumerated values.

Count null values: Select this check box to include null values as zeroes in summary fields.

CheckBox (Section Report)

In ActiveReports, you can use the CheckBox control to represent a Boolean value in a report. By default, it appears as a small box with text to the right. If the DataField value evaluates to True, the small box appears with a check mark; if False, the box is empty. By default, the checkbox is empty.

Important Properties

Property	Description
CheckAlignment	Gets or sets the alignment of the check box text within the control drawing area.
Checked	Gets or sets a value indicating whether the check box is in the checked state. You can also set the **Checked** property of the check box in code or bind it to a Boolean database value.
DataField	Gets or sets the field from the data source to bind to the control.
Text	Gets or sets the printed caption of the check box.

You can double-click in the CheckBox control to enter edit mode and enter text directly in the control, or you can enter text in the Properties window or you can assign data to display in code through the **Text** property.

You can format text in the CheckBox control in edit mode using the ActiveReports toolbar, or you can modify properties in the Properties window. Formats apply to all of the text in the control. Text formatting changes in the Properties window immediately appear in the control, and changes made in the toolbar are immediately reflected in the Properties window.

> **Note**: In edit mode for a CheckBox with the Alignment property set to Justify, the Alignment value temporarily changes to the default value, Left. Once you leave edit mode, it automatically changes back to Justify.

Keyboard Shortcuts

In edit mode, you can use the following keyboard shortcuts.

Key Combination	Action
Enter	New line.
Alt + Enter	Saves modifications and exits edit mode.
Esc	Cancels modifications and exits edit mode.

In the End User Designer, you can disable this feature in the **EditModeEntering ('EditModeEntering Event' in the on-line documentation)** and **EditModeExit ('EditModeExit Event' in the on-line documentation)** events.

CheckBox Dialog

With the control selected on the report, in the Commands section at the bottom of the Properties window, you can click the **Property dialog** command to open the dialog.

General

Name: Enter a name for the CheckBox that is unique within the report. This name is displayed in the Document Outline and in XML exports. You can only use underscore (_) as a special character in the Name field. Other special characters such as period (.), space ( ), forward slash (/), back slash (\), exclamation (!), and hyphen (-) are not supported.

Tag: Enter a string that you want to persist with the control. If you access this property in code, it is an object, but in the Properties window or Property dialog, it is a string.

Visible: Clear this check box to hide the control.

DataField: Select a field that returns a Boolean value from the data source to bind to the control. The value of this field determines how to set the Checked property at run time.

Text: Enter static text to show in the textbox. If you specify a DataField value, this property is ignored.

Check Alignment: Drop down the visual selector to choose the vertical and horizontal position for the check box within the control.

Checked: Select this check box to have the CheckBox control appear with a check mark in the box.

Appearance

Background Color: Select a color to use for the background of the textbox.

Font

Name: Select a font family name or a theme font.

Size: Choose the size in points for the font.

Style: Choose **Normal** or **Italic**.

Weight: Choose from **Normal** or **Bold**.

Color: Choose a color to use for the text.

Decoration: Select check boxes for **Underline** and **Strikeout**.

GDI Charset: Enter a value to indicate the GDI character set to use. For a list of valid values, see MSDN Font.GDICharSet Property.

GDI Vertical: Select this checkbox to indicate that the font is derived from a GDI vertical font.

Alignment

Wrap mode: Choose **NoWrap**, **WordWrap**, or **CharWrap** to select whether to wrap words or characters to the next line.

Padding

Enter values in points to set the amount of space to leave around the check box.

- **Top**
- **Left**
- **Right**
- **Bottom**

RichTextBox

In ActiveReports, the RichTextBox control is used to display, insert and manipulate formatted text. It is different from the TextBox control in a number of ways. The most obvious is that it allows you to apply different formatting to different parts of its content.

You can also load an RTF file or an HTML file into the RichTextBox control at design time or at run time, and you can use it to create field merged reports with field placeholders that you replace with values at run time. You can add fields to the text you enter directly in the control by right-clicking and choosing **Insert Fields** and providing a field name.

For more information, see Load a File into a RichTextBox Control and Mail Merge with RichTextBox.

Keyboard Shortcuts

In edit mode, you can use the following keyboard shortcuts.

Key Combination	Action
Enter	New line.
Alt + Enter	Saves modifications and exits edit mode.
Esc	Cancels modifications and exits edit mode.

In the End User Designer, you can disable this feature in the **EditModeEntering ('EditModeEntering Event' in the on-line documentation)** and **EditModeExit ('EditModeExit Event' in the on-line documentation)** events.

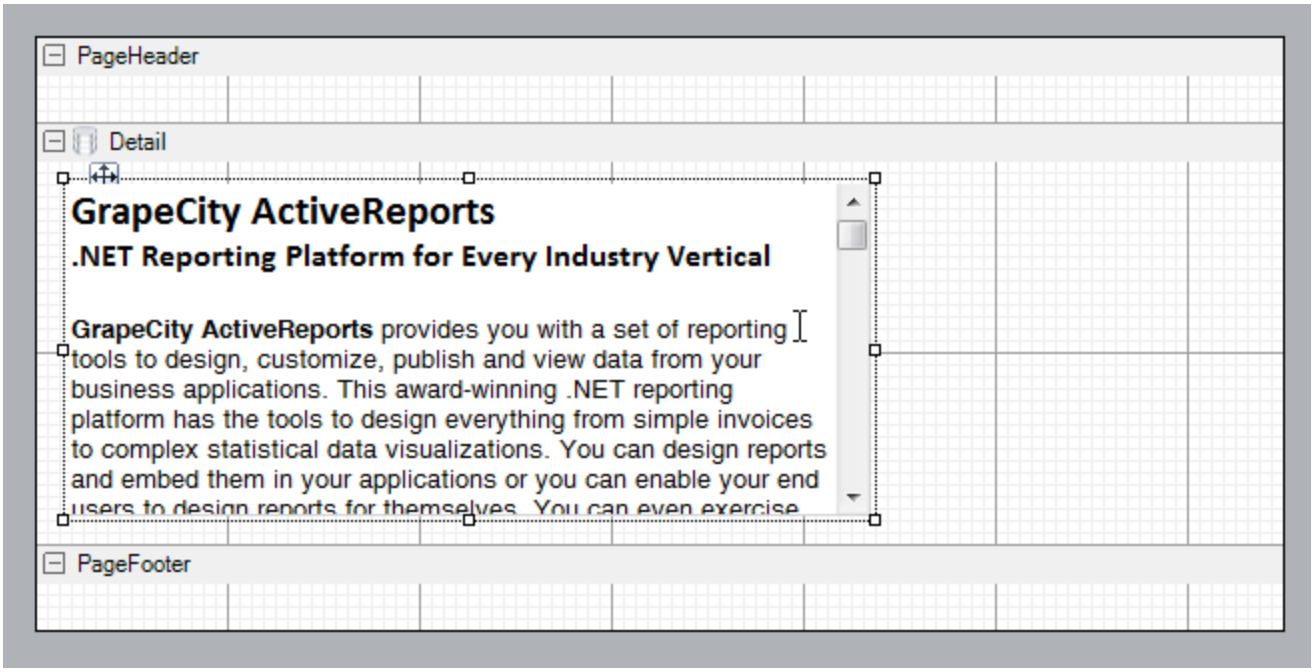
Load File Dialog

With the control selected on the report, in the Commands section at the bottom of the Properties window, you can click the **Load file** command to open the dialog. This allows you to select a file to load into the control at design time. Supported file types are as follows.

- Text (*.txt)
- RichText (*.rtf)
- HTML (*.htm, *.html)

Text/RichText file

When you load a Text/RichText file, the control will display the file at design time in the edit mode.

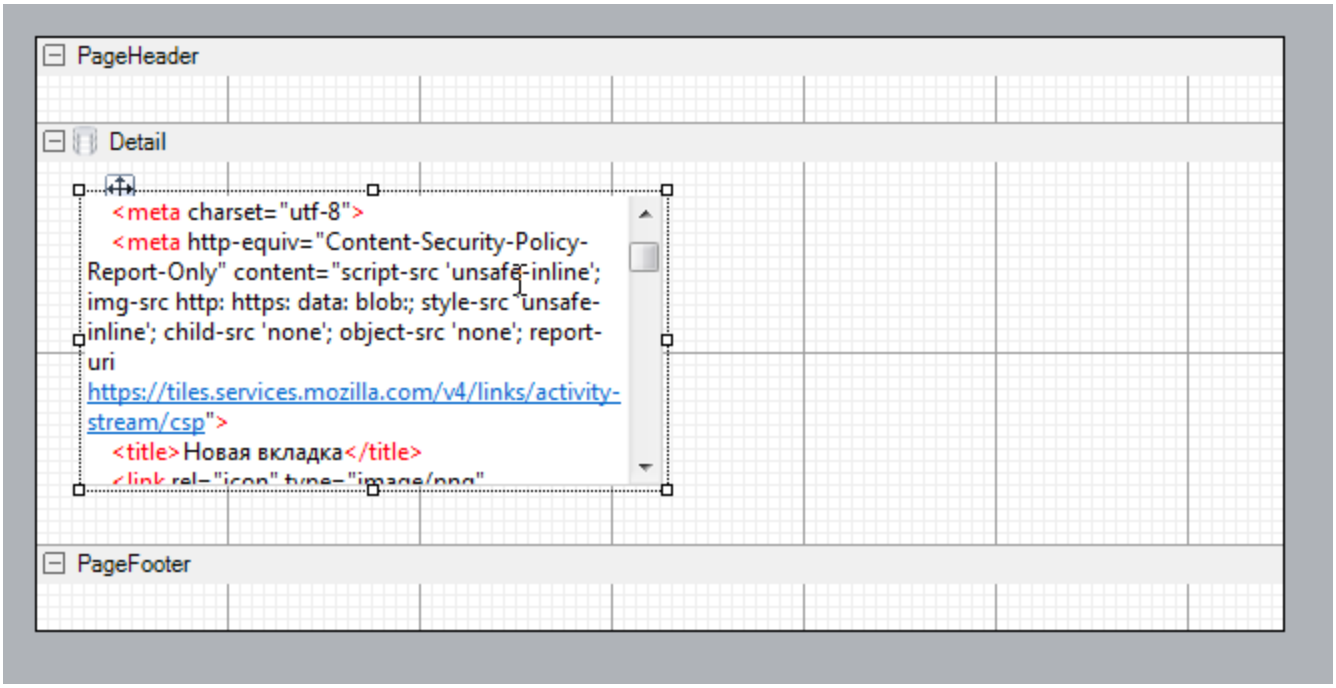


Important Properties

Property	Description
AutoReplaceFields	If True, any fields in the RTF control are replaced with fields from the data source.
CanGrow	Determines whether ActiveReports should increase the height of the control based on its content.
CanShrink	Determines whether ActiveReports should decrease the height of the field based on its value.
MultiLine	Gets or sets a value that determines whether the RichTextBox prints multiple lines or a single line.
DataField	Gets or sets the field name from the data source to bind to the control.

HTML File

When you load an HTML file, the control will display the HTML markup at design time in the edit mode.



⚠ **Important**: The HTML text in an HTML file, loaded into the RichTextBox control, must be enclosed in the **<body> </body>** tags. Otherwise, the HTML data is converted into RTF.

Supported Tags

See the **Supported HTML Tags** section in Formatted Text to learn about what HTML tags the RichTextBox control supports.

To load a file into the report at run time, use the Load method. For more information, see **Load Method (on-line documentation)**.

RichTextBox Dialog

With the control selected on the report, in the Commands section at the bottom of the Properties window, you can click the **Property dialog** command to open the dialog.

General

Name: Enter a name for the RichTextBox that is unique within the report. This name is displayed in the Document Outline. You can only use underscore (_) as a special character in the Name field. Other special characters such as period (.), space ( ), forward slash (/), back slash (\), exclamation (!), and hyphen (-) are not supported.

Tag: Enter a string that you want to persist with the control. If you access this property in code, it is an object, but in the Properties window or Property dialog, it is a string.

Visible: Clear this check box to hide the control.

DataField: Select a field from the data source to bind to the control.

Max length: Enter the maximum number of characters to display in the control. If you do not specify a value, it displays an unlimited number of characters.

AutoReplaceFields: Select this check box to have ActiveReports replace any fields in the control with values from the data

source.

Appearance

Background Color: Select a color to use for the background of the control.

Format

RichTextBox height

Can increase to accommodate contents: Select this check box to set CanGrow to True.

Can decrease to accommodate contents: Select this check box to set CanShrink to True.

Multiline: Select this check box to allow the control to display multiple lines of text.

Shape (Section Report)

In ActiveReports, the Shape control is used to add simple shapes to a report.

Important Properties

Property	Description
LineColor	Gets or sets the color of the shape lines.
LineStyle	Gets or sets the pen style used to draw the line.
LineWeight	Gets or sets the pen width used to draw the shape.
Style	Gets or sets the shape type to draw. You can select from Rectangle, Ellipse and a RoundedRect.
RoundingRadius	Sets the radius of each corner for the RoundRect shape type. You can select Default, TopLeft, TopRight, BottomLeft or BottomRight. Selecting Default sets the radius of all the corners of the Shape control to a specified percentage. Default value = 10 (percent).

Shape Dialog

With the control selected on the report, in the Commands section at the bottom of the Properties window, you can click the **Property dialog** command to open the dialog.

General

Name: Enter a name for the shape that is unique within the report. This name is displayed in the Document Outline and in XML exports. You can only use underscore (_) as a special character in the Name field. Other special characters such as period (.), space ( ), forward slash (/), back slash (\), exclamation (!), and hyphen (-) are not supported.

Tag: Enter a string that you want to persist with the control. If you access this property in code, it is an object, but in the Properties window or Property dialog, it is a string.

Visible: Clear this check box to hide the control.

Appearance

Shape type: Select the type of shape to display. You can choose from **Rectangle**, **Ellipse**, or **RoundRect**. For a circle, set the control width and height properties to the same value, and choose Ellipse, or choose RoundRect and set the Rounding

radius to 100%.

Rounded Rectangle: When the Shape type is set to **RoundRect**, you can specify the radius for each corner of the shape independently. Drag the handlers ◆ available at each corner of the shape to set the value of the radius at each corner.

> **Note:** To enable specific corners, check the CheckBox available near each corner of the Shape control.

- **Use the same radius on specified corners**: Select this option to apply the same radius to all selected corners of the shape.
- **Use different radius on specified corners:** Select this option to apply a different radius to each selected corner of the shape.

Line style: Select a line style to use for the shape line. You can set it to Transparent, Solid, Dash, Dot, DashDot, DashDotDot, or Double.

Line weight: Enter the width for the shape line.

Line color: Select a color to use for the shape line.

Background color: Select a color to use for the background of the shape.

Background style: Select a background style for the shape from Solid, Gradient, and Pattern. Depending on the style selected, other properties are available.

Picture

In section reports, the Picture control is used to print an image on the report. In the **Image** property of the Picture control, you can select any image file to display on your report.

> **Note**: Use the **PictureAlignment** and **SizeMode** properties to control cropping and alignment.

Important Properties

Property	Description
LineColor	Gets or sets the border line color around the Picture control.
LineStyle	Gets or sets the pen style used to paint the border around the Picture control.
LineWeight	Gets or sets the pen width of the border line.
PictureAlignment	Gets or sets the position of the image within the control area.
Description	Gets or sets the alternate description for the picture. Used in the Html Export for the "alt" img tag property.
HyperLink	Gets or sets a URL address that can be used in the viewer's Hyperlink event to navigate to the specified location. The URL is automatically converted into an anchor tag or a hyperlink in HTML and PDF exports.
Image	Gets or sets the image to print.
SizeMode	Gets or sets a value that determines how the image is sized to fit the Picture control area.

Picture Dialog

With the control selected on the report, in the Commands section at the bottom of the Properties window, you can click the **Property dialog** command to open the dialog.

General

Name: Enter a name for the picture control that is unique within the report. This name is displayed in the Document Outline and in XML exports. You can only use underscore (_) as a special character in the Name field. Other special characters such as period (.), space ( ), forward slash (/), back slash (\), exclamation (!), and hyphen (-) are not supported.

Tag: Enter a string that you want to persist with the control. If you access this property in code, it is an object, but in the Properties window or Property dialog, it is a string.

Visible: Clear this check box to hide the control.

DataField: Select a field from the data source to bind to the control.

Choose image: Click this button open a dialog where you can navigate to a folder from which to select an image file to display.

HyperLink: Enter a URL to use in the Viewer HyperLink event. The URL automatically converts to an anchor tag or hyperlink in PDF and HTML exports.

Title: Enter static text for the picture.

Description: Enter text to describe the image for those who cannot see it. This is used in the HTML export for the "alt" attribute of the img tag.

Appearance

Line style: Select a line style to use for the border line. You can set it to Transparent, Solid, Dash, Dot, DashDot, DashDotDot, or Double.

Line weight: Enter the width for the border line.

Line color: Select a color to use for the border line.

Background color: Select a color to use for the background of the picture control.

Picture alignment: Select how to align the image within the control. You can select from TopLeft, TopRight, Center, BottomLeft, or BottomRight.

Size mode: Select how to size the image within the control. You can select from Clip, Stretch, or Zoom. Clip uses the original image size and clips off any excess, Stretch fits the image to the size and shape of the control, and Zoom fits the image into the control while maintaining the aspect ratio of the original image.

Line (Section Report)

In ActiveReports, the Line control allows you to draw vertical, horizontal or diagonal lines that visually separate or highlight areas within a section on a report.

> **Note**: If you need lines to span across report sections, please see the CrossSectionLine control.

You can use your mouse to visually move and resize the Line, or you can use the Properties window to change its **X1**, **X2**,

Y1, and **Y2** properties to specify the coordinates for its starting and ending points.

Important Properties

Property	Description
AnchorBottom	Anchors the line to the bottom of the containing section so that the line grows along with the section.
LineColor	Gets or sets the color of the line.
LineStyle	Gets or sets the pen style used to draw the line.
LineWeight	Gets or sets the pen width of the line.
X1	Gets or sets the horizontal coordinate of the line's starting point.
X2	Gets or sets the horizontal coordinate of the line's end point.
Y1	Gets or sets the vertical coordinate of the line's starting point.
Y2	Gets or sets the vertical coordinate of the line's end point.

Line Dialog

With the control selected on the report, in the Commands section at the bottom of the Properties window, you can click the **Property dialog** command to open the dialog.

General

Name: Enter a name for the line that is unique within the report. This name is displayed in the Document Outline and in XML exports. You can only use underscore (_) as a special character in the Name field. Other special characters such as period (.), space ( ), forward slash (/), back slash (\), exclamation (!), and hyphen (-) are not supported.

Tag: Enter a string that you want to persist with the control. If you access this property in code, it is an object, but in the Properties window or Property dialog, it is a string.

Visible: Clear this check box to hide the control.

Appearance

Line style: Select a line style to use for the line. You can set it to Transparent, Solid, Dash, Dot, DashDot, DashDotDot, or Double.

Line weight: Enter the width for the line.

Line color: Select a color to use for the line.

Anchor at bottom: Select this check box to automatically change the Y2 value to the value of the bottom edge of the containing section after it has grown to accommodate data at run time.

Page Break

You can cause ActiveReports to break to a new page at any point within any section using the PageBreak control. All controls placed below the PageBreak in the section render to a new page.

> ⚠ **Caution:** It is not recommended to place PageReport control over other controls as it can cause extra page breaks.

> **Tip**: Another way to cause ActiveReports to break to a new page is by setting the **NewPage** property of a section to Before, After, or BeforeAfter. This property is available on any section except for PageHeader and PageFooter.

Important Properties

Property	Description
Enabled	Determines whether to enable the PageBreak.
Location - X	Gets or sets the horizontal location of an object.
Location - Y	Gets or sets the vertical location of an object.

PageBreak Dialog

With the control selected on the report, in the Commands section at the bottom of the Properties window, you can click the **Property dialog** command to open the dialog.

General

Name: Enter a name for the PageBreak that is unique within the report. This name is displayed in the Document Outline and in XML exports. You can only use underscore (_) as a special character in the Name field. Other special characters such as period (.), space ( ), forward slash (/), back slash (\), exclamation (!), and hyphen (-) are not supported.

Tag: Enter a string that you want to persist with the control. If you access this property in code, it is an object, but in the Properties window or Property dialog, it is a string.

Enabled: Select a field from the data source to bind to the control.

Barcode (Section Report)

The **Barcode** report control offers various barcode styles to choose from. This saves you the time and expense of finding and integrating a separate component. As with other data-bound report controls, you can bind a barcode to data using the **DataField** property.

Apart from the barcode style, you can manage the alignment, color, background color, caption position, font, text, and check whether checksum is enabled in the Properties Window. There are more properties available with the Code49, PDF417, and QRCode barcode styles. Click the Barcode to reveal its properties in the Properties window. All of the properties specific to this report control are also available in the Barcode dialog.

Important Properties

The following properties help you to customize the specific barcode you need for your application:

Property	Description
Alignment	The horizontal alignment of the caption in the control. Select from Near, Center, or Far. See CaptionPosition for vertical alignment.
AutoSize	When set to True, the barcode automatically stretches to fit the control.
BackColor	Select a background fill color for the barcode.
BarHeight	Set the height, in inches, of the barcode's bars. If the bar height exceeds the height of the control, this property is ignored.

CaptionGrouping	Gets or sets a value indicating whether to add spaces between groups of characters in the caption to make long numbers easier to read. This property is only available with certain styles of barcode, and is ignored with other styles.
CaptionPosition	The vertical alignment of the caption in the control. Select from None, Above, or Below. See Alignment for horizontal alignment. None is selected by default, and no caption is displayed.
CheckSumEnabled	Some barcode styles require a checksum and some have an optional checksum. CheckSumEnabled has no effect if the style already requires a check digit or if the style does not offer a checksum option.
Font	Set the font for the caption. Only takes effect if you set the CaptionPosition property to a value other than None.
ForeColor	Select a color for the barcode and caption.
NarrowBarWidth	Also known as the X dimension, this is a value defining the width of the narrowest part of the barcode. Before using an extremely small value for this width, ensure that the scanner can read it. This value is specified in pixels (for example, 10 pixels).
NWRatio	Also known as the N dimension, this is a value defining the multiple of the ratio between the narrow and wide bars in symbologies that contain bars in only two widths. For example, if it is a 3 to 1 ratio, this value is 3.
QuietZone	Sets an area of blank space on each side of a barcode that tells the scanner where the symbology starts and stops. You can set separate values for the Left, Right, Top, and Bottom.
Rotation	Sets the amount of rotation to use for the barcode. You can select from None, Rotate90Degrees, Rotate180Degrees, or Rotate270Degrees.
Style	Sets the symbology used to render the barcode. See the table below for details about each style.
SupplementOptions	Sets the 2/5-digit add-ons for EAN/UPC symbologies. You can specify Text, DataField, BarHeight, CaptionPosition, and Spacing for the supplement.
Text	Sets the value to print as a barcode symbol and caption. ActiveReports fills this value from the bound data field if the control is bound to the data source.

Limitations

Some barcode types may render incorrectly and contain white lines in the Html and RawHtml views. However, this limitation does not affect printing and scanning. The list of barcode types that may render with white lines in the Html and RawHtml views:

- Code49
- QRCode
- Pdf417
- RSSExpandedStacked
- RSS14Stacked
- RSS14StackedOmnidirectional

Barcode Dialog

With the control selected on the report, in the Commands section at the bottom of the Properties window, you can click

the **Property dialog** command to open the dialog.

General

Name: Enter a name for the control that is unique within the report. This name is displayed in the Document Outline and in XML exports. You can only use underscore (_) as a special character in the Name field. Other special characters such as period (.), space ( ), forward slash (/), back slash (\), exclamation (!), and hyphen (-) are not supported.

Tag: Enter a string that you want to persist with the control. If you access this property in code, it is an object, but in the Properties window or Property dialog, it is a string.

Visible: Clear this check box to hide the control.

DataField: Select a field from the data source to bind to the control.

Text: Enter static text to show in the textbox. If you specify a DataField value, this property is ignored.

Autosize: Clear this check box to prevent the barcode from automatically resizing to fit the control.

Caption

Location: Select a value to indicate whether and where to display a caption for the barcode. You can select from Above, Below, or None.

Text alignment: Select a value to indicate how to align the caption text. You can select from Center, Near, or Far.

Barcode Settings

Style: Enter the type of barcode to use. ActiveReports supports all of the most popular symbologies:

Table of all included symbologies

> **Notes**: The RSS and QRCode styles have fixed height-to-width ratios. When you resize the width, the height is automatically calculated.
>
> When you choose a style that offers supplemental options, the additional options appear below.

Symbology Name	Example	Description
Ansi39		ANSI 3 of 9 (Code 39) uses upper case, numbers, - , * $ / + %. This is the default barcode style.
Ansi39x		ANSI Extended 3 of 9 (Extended Code 39) uses the complete ASCII character set.
BC412		Data BC412 uses 35 characters, 0 - 9 and A - Z. It is designed for semiconductor wafer identification.
Codabar		Codabar uses A B C D + - : . / $ and numbers.

Code_11		Encodes the numbers 0 through 9, the hyphen (-), and start/stop characters. It is primarily used in labeling telecommunications equipment.
Code_128_A		**Code 128 A** uses control characters, numbers, punctuation, and upper case.
Code_128_B		**Code 128 B** uses punctuation, numbers, upper case and lower case.
Code_128_C		**Code 128 C** uses only numbers.
Code_128auto		Code 128 Auto uses the complete ASCII character set. Automatically selects between Code 128 A, B and C to give the smallest barcode.
Code_2_of_5		Code 2 of 5 uses only numbers.
Code_93		Code 93 uses uppercase, % $ * / , + -, and numbers.
Code25intlv		Interleaved 2 of 5 uses only numbers.
Code39		Code 39 uses numbers, % * $ /. , - +, and upper case.
Code39x		Extended Code 39 uses the complete ASCII character set.
Code49		**Code 49** is a 2D high-density stacked barcode containing two to eight rows of eight characters each. Each row has a start code and a stop code. Encodes the complete ASCII character set.

Code93x		Extended Code 93 uses the complete ASCII character set.
DataMatrix		**Data Matrix** is a high density, two-dimensional barcode with square modules arranged in a square or rectangular matrix pattern.
EAN_13		**EAN-13** uses only numbers (12 numbers and a check digit). It takes only 12 numbers as a string to calculate a check digit (CheckSum) and add it to the thirteenth position. The check digit is an additional digit used to verify that a bar code has been scanned correctly. The check digit is added automatically when the CheckSumEnabled property is set to True.
EAN_13 with the add-on code		**EAN-13** may include the add-on code to the right of the main code. The add-on code may include up to 5 supplemental characters.
EAN_8		**EAN-8** uses only numbers (7 numbers and a check digit).
EAN128FNC1		**EAN-128** is an alphanumeric one-dimensional representation of Application Identifier (AI) data for marking containers in the shipping industry. This barcode is now obsolete. You should use UCC/EAN–128 instead which provides similar functionality with better performance.
GS1QRCode		GS1QRCode is a subset of the QR Code. The GS1 QR Code is a 2D symbol that denotes the Extended Packaging URL for a trade item. It is processed to obtain one URL address associated with the trade item identified by the Global Trade Item Number (GTIN). GS1 QR Code requires the mandatory association of the GTIN and Extended Packaging URL. GS1 QR Code allows to encode GS1 System

		Application Identifiers (AI) into QR Code 2D barcodes. **Limitation:** Kanji, CN, JP and Korean characters.
HIBCCode128		HIBCCode128 barcode uses the Code128 symbology. It encodes Primary Data and Secondary Data using slash (/) as delimiter. It is used in the health care products industry for identification purpose.
HIBCCode39		HIBCCode39 barcode uses the Code39 symbology, with the Code39OptionalCheckDigit property set to True. It encodes 'Primary Data' and 'Secondary Data' using slash (/) as delimiter. It is used in the health care products industry for identification purpose.
IATA_2_of_5		IATA_2_of_5 is a variant of Code_2_of_5 and uses only numbers with a check digit.
IntelligentMail		Intelligent Mail, formerly known as the 4-State Customer Barcode, is a 65-bar code used for domestic mail in the U.S.
IntelligentMailPackage		IntelligentMailPackage is more efficient in terms of processing and tracking mails than Intelligent Mail barcode.
ISBN		International Standard Book Number barcode is a special form of the EAN-13 code and is used as a unique 9-digit commercial book identifier.
ISMN		Internationally Standard Music Number barcode is a special form of the EAN-13 code. It is used for marking printed musical publications.
ISSN		International Standard Serial Number barcode is a special form of the EAN-13 code. It is used to identify serial publications, publications that are issued in numerical order, such as the volumes of a magazine.

ITF14		Interleaved Two of Five This code is used to mark cartons and palettes that are including goods with an EAN-13 code. One digit is added in front of the EAN-13 code to mark the packing variant.
JapanesePostal		This is the barcode used by the Japanese Postal system. Encodes alpha and numeric characters consisting of 18 digits including a 7-digit postal code number, optionally followed by block and house number information. The data to be encoded can include hyphens.
Matrix_2_of_5		Matrix 2 of 5 is a higher density barcode consisting of 3 black bars and 2 white bars.
MaxiCode		MaxiCode is special polar barcode that uses 256 characters. It is used to encode a specific amount of data.
MicroPDF417		**MicroPDF417** is two-dimensional (2D), multi-row symbology, derived from PDF417. Micro-PDF417 is designed for applications that need to encode data in a two-dimensional (2D) symbol (up to 150 bytes, 250 alphanumeric characters, or 366 numeric digits) with the minimal symbol size. MicroPDF417 allows you to insert an FNC1 character as a field separator for variable length Application Identifiers (AIs). To insert FNC1 character, set “\n” for C#, or “vbLf” for VB to Text property at run time.
MicroQRCode		**MicroQRCode** is a two-dimensional (2D) barcode that is designed for applications that use a small amount of data. It can handle numeric and alphanumeric data as well as Japanese kanji and kana characters. This symbology can encode up to 35 numeric characters.

MSI		MSI Code uses only numbers.
Pdf417		Pdf417 is a popular high-density 2-dimensional symbology that encodes up to 1108 bytes of information. This barcode consists of a stacked set of smaller barcodes. Encodes the full ASCII character set. It has ten error correction levels and three data compaction modes: Text, Byte, and Numeric. This symbology can encode up to 1,850 alphanumeric characters or 2,710 numeric characters.
Pharmacode		Pharmacode represents only numeric data from 3 to 131070. It is a barcode standard used in the pharmaceutical industry for packaging. It is designed to be readable despite printing errors.
Plessey		Plessey uses hexadecimal digits to encode. It is a one-dimensional barcode used mainly in libraries.
PostNet		PostNet uses only numbers with a check digit.
PZN		PZN Pharmaceutical Central/General Number uses the same encoding algorithm as Code 39 but can carry only digits – 0123456789. The number of digits supported for encoding are 6 or 7. The letters 'PZN' and checksum digit are automatically added. It is mainly used to identify medicine and health-care products in Germany and other German-speaking countries.
QRCode		**QRCode** is a 2D symbology that is capable of handling numeric, alphanumeric and byte data as well as Japanese kanji and kana characters. This symbology can encode up to 7,366 characters.

RM4SCC		Royal Mail RM4SCC uses only letters and numbers (with a check digit). This is the barcode used by the Royal Mail in the United Kingdom.
RSS14		RSS14 is a 14-digit Reduced Space Symbology that uses EAN.UCC item identification for point-of-sale omnidirectional scanning. The RSS family of barcodes is also known as **GS1 DataBar**.
RSS14Stacked		RSS14Stacked uses the EAN.UCC information with Indicator digits as in the RSS14Truncated, but stacked in two rows for a smaller width. RSS14Stacked allows you to set Composite Options, where you can select the type of the barcode in the **Type** drop-down list and the value of the composite barcode in the **Value** field.
RSS14Stacked CCA		RSS14Stacked with Composite Component - Version A.
RSS14StackedOmnidirectional		RSS14StackedOmnidirectional uses the EAN.UCC information with omnidirectional scanning as in the RSS14, but stacked in two rows for a smaller width.
RSS14Truncated		RSS14Truncated uses the EAN.UCC information as in the RSS14, but also includes Indicator digits of zero or one for use on small items not scanned at the point of sale.
RSSExpanded		RSSExpanded uses the EAN.UCC information as in the RSS14, but also adds AI elements such as weight and best-before dates. RSSExpanded allows you to insert an FNC1 character as a field separator for variable length Application Identifiers (AIs).

		To insert FNC1 character, set “\n” for C#, or “vbLf” for VB to Text property at run time.
RSSExpandedStacked		**RssExpandedStacked** uses the EAN.UCC information with AI elements as in the RSSExpanded, but stacked in two rows for a smaller width. RSSExpandedStacked allows you to insert an FNC1 character as a field separator for variable length Application Identifiers (AIs). To insert FNC1 character, set “\n” for C#, or “vbLf” for VB to Text property at run time.
RSSLimited		RSS Limited uses the EAN.UCC information as in the RSS14, but also includes Indicator digits of zero or one for use on small items not scanned at the point of sale. RSSLimited allows you to set Composite Options, where you can select the type of the barcode in the **Type** drop-down list and the value of the composite barcode in the **Value** field.
RSSLimited CCA		RSS Limited with Composite Component - Version A.
SSCC_18		SSCC_18 is an 18-digit Serial Shipping Container Code. It is used to identify individual shipping containers for tracking purposes.
Telepen		Telepen has 2 different modes - alphanumeric-only and numeric-only. Both modes require a start character, a check digit, and a stop character. It is mainly used in manufacturing industries.
UCCEAN128		UCC/EAN –128 complies to GS1-128 standards. GS1-128 uses a series of Application Identifiers to encode data. This barcode uses the complete ASCII character set. It also uses FNC1 character as the first character position. Using AI's, it encodes best before dates, batch numbers, weights, and more such attributes. It is also used in HIBC applications.

UPC_A		**UPC-A** uses only numbers (11 numbers and a check digit).
UPC_A with the add-on code		**UPC-A** may include the add-on code to the right of the main code. The add-on code may include up to 5 supplemental characters.
UPC_E0		**UPC-E0** uses only numbers. Used for zero-compression UPC symbols. For the Caption property, you may enter either a six-digit UPC-E code or a complete 11-digit (includes code type, which must be zero) UPC-A code. If an 11-digit code is entered, the Barcode control will convert it to a six-digit UPC-E code, if possible. If it is not possible to convert from the 11-digit code to the six-digit code, nothing is displayed.
UPC_E0 with the add-on code		**UPC-E0** may include the add-on code to the right of the main code. The add-on code may include up to 5 supplemental characters.
UPC_E1		**UPC-E1** uses only numbers. Used typically for shelf labeling in the retail environment. The length of the input string for U.P.C. E1 is six numeric characters.
UPC_E1 with the add-on code		**UPC-E1** may include the add-on code to the right of the main code. The add-on code may include up to 5 supplemental characters.

Bar Height: Enter a value in inches (for example, .25in) for the height of the barcode.

Narrow Bar Width (also known as X dimension): Enter a value in points (for example, 0.8pt) for the width of the narrowest part of the barcode. Before using an extremely small value for this width, ensure that the scanner can read it.

Tip: For accurate scanning, the quiet zone should be ten times the Narrow Bar Width value.

Narrow Width Bar Ratio: Enter a value to define the multiple of the ratio between the narrow and wide bars in symbologies that contain bars in only two widths. For example, if it is a 3 to 1 ratio, this value is 3. Commonly used values are 2, 2.5, 2.75, and 3.

QuietZone

A quiet zone is an area of blank space on either side of a barcode that tells the scanner where the symbology starts and stops.

Left: Enter a size in inches of blank space to leave to the left of the barcode.

Right: Enter a size in inches of blank space to leave to the right of the barcode.

Top: Enter a size in inches of blank space to leave at the top of the barcode.

Bottom: Enter a size in inches of blank space to leave at the bottom of the barcode.

Note: The units of measure listed for all of these properties are the default units of measure used if you do not specify. You may also specify **cm**, **mm**, **in**, **pt**, or **pc**.

Checksum

A checksum provides greater accuracy for many barcode symbologies.

Compute Checksum: Select whether to automatically calculate a checksum for the barcode.

Note: If the symbology you choose requires a checksum, setting this value to **False** has no effect.

Code49 Options

Code49 Options are available for the Code49 barcode style.

Grouping: Indicates whether to use grouping for the Code49 barcode. The possible values are **True** or **False** (default). If Grouping is set to True, any value not expressed by a single barcode is expressed by splitting it into several barcodes.

GroupNumber: Enter a number between 0 (default) and 8 for the barcode grouping. When the Group property is set to 2, the grouped barcode's second symbol is created. When invalid group numbers are set, the BarCodeDataException is thrown.

Code128 Options

Code128 has three settings that work in conjunction: Dpi, BarAdjust, and ModuleSize. This property only applies to the barcode style EANFNC1. You can improve the readability of the barcode by setting all three properties.

- **Dpi**: Sets the printer resolution. Specify the resolution of the printer as dots per inch to create an optimized barcode image with the specified Dpi value.
- **BarAdjust**: Sets the adjustment size by dot units, which affects the size of the module and not the entire barcode.
- **ModuleSize**: Sets the horizontal size of the barcode module.

DataMatrix Options

DataMatrix Options are available for the DataMatrix barcode style.

EccMode: Select the Ecc mode from the drop-down list. The possible values are **ECC000**, **ECC050**, **ECC080**, **ECC100**, **ECC140** or **ECC200**.

Ecc200 Symbol Size: Select the size of the ECC200 symbol from the drop-down list. The default value is **SquareAuto**.

Ecc200 Encoding Mode: Select the encoding mode for ECC200 from the drop-down list. The possible values are **Auto**, **ASCII**, **C40**, **Text**, **X12**, **EDIFACT** or **Base256**.

Ecc000_140 Symbol Size: Select the size of the ECC000_140 barcode symbol from the drop-down list.

Structured Append: Select whether the barcode symbol is part of the structured append symbols. The possible values are **True** or **False**.

Structure Number: Enter the structure number of the barcode symbol within the structured append symbols.

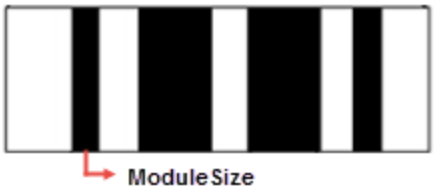
File Identifier: Enter the file identifier of a related group of the structured append symbols. If you set the value to 0, the file identifier symbols are calculated automatically.

EAN128FNC1 Options

EAN128FNC1 Options are available for the EAN128FNC1 barcode style.

DPI: Specify the printer resolution.

Module Size: Enter the horizontal size of the barcode module.



Bar Adjust: Enter the adjustment size by dot units, which affects the size of the module and not the entire barcode.

GS1Composite Options

GS1Composite Options are available for the RSS14Stacked and RSSLimited barcode styles.

Type: Select the type of the composite barcode from the drop-down list. The possible values are **None** or **CCA**. CCA (Composite Component - Version A) is the smallest variant of the 2-dimensional composite component.

Value: Enter the expression to set the value of the composite barcode.

MaxiCode Options

MaxiCode option to select mode is available for MaxiCode barcode.

Mode: Select the mode of the MaxiCode barcode. The available values are Mode2 to Mode6.

MicroPDF417 Options

MicroPDF417 Options are available for the MicroPDF417 barcode style.

Compaction Mode: Select the type of the compaction mode from the drop-down list. The possible values are **Auto**, **TextCompactionMode**, **NumericCompactionMode,** or **ByteCompactionMode**.

Version: Select the version from the drop-down box to set the symbol size.

Segment Index: The segment index of the structured append symbol. The valid value is from 0 to 99998, and less than the value in Segment Count.

Segment Count: The segment count of the structured append symbol. The valid value is from 0 to 99999.

File ID: The file id of the structured append symbol. The valid value is from 0 to 899.

MicroQRCode Options

MicroQRCode Options are available for the MicroQRCode barcode style.

ErrorLevel: Select the error correction level for the barcode from the drop-down list. Valid values are **M**, **L**, or **Q**. The available Error Level values change depending on the version you select.

Version: Enter the version of the MicroQRCode barcode style. Valid values are **M1**, **M2**, **M3**, or **M4.** The maximum amount of data can be stored in version M4.

Mask: Select the pattern for the barcode masking from the drop-down list. Valid values are **Mask00**, **Mask01**, **Mask10**, or **Mask11**.

Encoding: Select the barcode encoding from the drop-down list.

PDF417 Options

PDF417 Options are available for the Pdf417 barcode style.

Columns: Enter column numbers for the barcode. Values for this property range from 1 to 30. The default value is -1 which automatically determines column numbers.

Rows: Enter row numbers for the barcode. Values range between 3 and 90. The default value is -1 which automatically determines row numbers.

ErrorLevel: Enter the error correction level for the barcode. Values range between 0 and 8. The error correction capability increases as the value increases. With each increase in the ErrorLevel value, the size of the barcode increases. The default value is -1 for automatic configuration.

PDF 417 Barcode Type: Select the PDF417 barcode type form the drop-down list. The possible values are **Normal** or **Simple**. Simple is the compact type in which the right indicator is neither displayed nor printed.

QRCode Options

QRCode Options are available for the QRCode barcode style.

Model: Select the model for the QRCode barcode style from the drop-down list. The possible values are **Model1**, the original model or **Model2**, the extended model. For **GS1QRCode**, Model1 is not supported.

ErrorLevel: Select the error correction level for the barcode from the drop-down list. The possible values are **L** (7% restorable), **M** (15% restorable), **Q** (25% restorable), and **H** (30% restorable). The higher the percentage, the larger the barcode becomes.

Version: Enter the version of the QRCode barcode style. Version indicates the size of the barcode. As the value increases, the barcode's size increases, enabling more information to be stored. Specify any value between 1 and 14 when the Model property is set to Model1 and 1 to 40 for Model2. The default value is -1, which automatically determines the version most suited to the value.

Mask: Select the pattern for the barcode masking form the drop-down list. Mask is used to balance brightness and offers 8 patterns in the QRCodeMask enumeration. The default value is Auto, which sets the masking pattern automatically, and is recommended for most uses.

- Mask000 (i+j) mod 2 = 0
- Mask001 i mod 2 = 0
- Mask010 j mod 3 = 0
- Mask011 (i+j) mod 3 = 0
- Mask100 ((i div 2)+(j div 3)) mod 2 = 0
- Mask101 (ij) mod 2 + (ij) mod 3 = 0
- Mask110 ((ij) mod 2 +(ij) mod 3) mod 2 = 0
- Mask111 ((ij)mod 3 + (i+j) mod 2) mod 2 = 0

Connection: Select whether to use the connection for the barcode. The possible values are **True** or **False**. This property is used in conjunction with the ConnectionNumber property. This property is not applicable to GS1QRCode barcode.

ConnectionNumber: Enter the connection number for the barcode. Use this property with the Connection property to set the number of barcodes it can split into. Values between 0 and 15 are valid. An invalid number raises the BarCodeData Exception. This property is not applicable to GS1QRCode barcode.

Encoding: Select the barcode encoding from the drop-down list.

RssExpandedStacked Options

RssExpandedStacked Options are available for the RSSExpandedStacked barcode style.

Row Count: Enter the number of the barcode stacked rows.

Supplementary Options

Supplementary Options are available for UPC_A, UPC_E0, UPC_E1, EAN_13, and EAN_8 barcode styles.

Supplement DataField: Select the data field for the barcode supplement.

Supplement Value: Enter the expression to set the value of the barcode supplement.

Caption Location: Select the location for the supplement caption from the drop-down list. The possible values are **None**, **Above** or **Below**.

Supplement Bar Height: Enter the bar height for the barcode supplement.

Supplement Spacing: Enter the spacing between the main and supplement barcodes.

Appearance

Fore color: Select a color to use for the bars in the barcode.

Background color: Select a color to use for the background of the control.

Rotation: Select a value indicating the degree of rotation to apply to the barcode. You can select from None, Rotate90Degrees, Rotate180Degrees, or Rotate270Degrees.

Font

Name: Select a font family name to use for the caption.

Size: Choose the size in points for the font.

Style: Choose from **Normal** or **Italic**.

Weight: Choose from **Normal** or **Bold**.

Decoration: Select check boxes for **Underline** and **Strikeout**.

GDI Charset: Enter a value to indicate the GDI character set to use. For a list of valid values, see MSDN Font.GDICharSet Property.

GDI Vertical: Select this checkbox to indicate that the font is derived from a GDI vertical font.

SubReport (Section Report)

In section reports, you can use the SubReport control to embed a report into another report. Once you place the Subreport control on a report, use code to create an instance of the report you want to load in it, and to attach the report object to the SubReport.

You can also pass parameters to the subreport from the main report so that data related to the main report displays in each instance of the subreport.

When to use a subreport

Due to the high overhead of running a second report and embedding it in the first, it is generally best to consider whether you need to use subreports. Some good reasons to use subreports include:

- Multiple data sources
- Multiple detail sections
- Side-by-side charts or tables

Remove page-dependent features from reports to be used as subreports

Subreports are disconnected from any concept of a printed page because they render inside the main report. For this reason, page-dependent features are not supported for use in subreports. Keep any such logic in the main report. Page-related concepts that are not supported in subreports include:

- Page numbers
- Page header and footer sections (delete these sections to save processing time)
- KeepTogether properties
- GroupKeepTogether properties
- NewPage properties

Coding best practices

Use the **ReportStart** event of the main report to create an instance of the report for your SubReport control, and then dispose of it in the **ReportEnd** event. This way, you are creating only one subreport instance when you run the main report.

In the **Format** event of the containing section, use the **Report** property of the SubReport control to attach a report object to the SubReport control.

> ⚠ **Caution**: It is not a recommended practice to initialize the subreport in the **Format** event. Doing so creates a new instance of the subreport each time the section processes. This consumes a lot of memory and processing time, especially in a report that processes a large amount of data.

Important Properties

Property	Description
CanGrow	Determines whether ActiveReports increases the height of the control based on its content.
CanShrink	Determines whether ActiveReports decreases the height of the control based on its value.
CloseBorder	By default, the bottom border of the control does not render until the end of the subreport. Set this property to True to have it render at the bottom of each page. (Only available in code.)
Report	Attaches a report object to the control. (Only available in code.)

SubReport Dialog

With the control selected on the report, in the Commands section at the bottom of the Properties window, you can click the **Property dialog** command to open the dialog.

General

Name: Enter a name for the SubReport that is unique within the report. This name is displayed in the Document Outline and in XML exports. You can only use underscore (_) as a special character in the Name field. Other special characters such as period (.), space ( ), forward slash (/), back slash (\), exclamation (!), and hyphen (-) are not supported.

Tag: Enter a string that you want to persist with the control. If you access this property in code, it is an object, but in the Properties window or Property dialog, it is a string.

Visible: Clear this check box to hide the control.

ReportName: Enter the name of the report.

Format

SubReport Height

Can increase to accommodate contents: Clear this check box to set CanGrow to False.

Can decrease to accommodate contents: Clear this check box to set CanShrink to False.

Ole Object

The OleObject control is hidden from the toolbox by default, and is only retained for backward compatibility. You can

enable the OleObject control in the Visual Studio toolbox only.

To enable the control in the Visual Studio toolbox, you must change the **EnableOleObject** property to **true**. This property can be found here: `C:\Program Files (x86)\GrapeCity\ActiveReports 14\Grapecity.ActiveReports.config`

Once enabled, you can add the OleObject control to reports. When you drop the control onto your report, the Insert Object dialog appears. This dialog allows you to create a new object or select one from an existing file.

Note: When you deploy reports that use the OleObject, you must also deploy the GrapeCity.ActiveReports.Interop.dll.

Caution: The WPF Viewer does not support the OLE object. If you preview a report containing the OLE object in the WPF Viewer, the OLE object will not be displayed.

Important Properties

Property	Description
PictureAlignment	Gets or sets the position of the object's content within the control area.
Class	Specifies the class name of the Ole object.
SizeMode	Gets or sets a value that determines how the object is sized to fit the OleObject control area.

Insert Object Dialog

The **Insert Object** dialog provides the following two options:

- **Create New** lets you select from a list of object types that you can insert into your report.

  Object Types

  - Adobe Acrobat Document
  - Microsoft Equation 3.0
  - Microsoft Excel 97-2003 Worksheet
  - Microsoft excel Binary Worksheet
  - Microsoft Excel Chart
  - Microsoft Excel Macro-Enabled Worksheet
  - Microsoft Excel Worksheet
  - Microsoft Graph Chart
  - Microsoft PowerPoint 97-2003 Presentation
  - Microsoft PowerPoint 97-2003 Slide
  - Microsoft PowerPoint Macro-Enabled Presentation
  - Microsoft PowerPoint Macro-Enabled Slide
  - Microsoft PowerPoint Presentation
  - Microsoft PowerPoint Slide
  - Microsoft Word 97-2003 Document
  - Microsoft Word Document
  - Microsoft Macro-Enabled Document
  - OpenDocument Presentation
  - OpenDocument Spreadsheet
  - OpenDocument Text
  - Package
  - Paintbrush Picture
  - Wordpad Document
- **Create from File** allows you to insert the contents of the file as an object into your document so that you can display it while printing.

Chart

In ActiveReports, you can use the **Chart** data region to present data graphically in a report. The chart offers you 17 core

chart types along with all of their variations, plus access to properties that control every aspect of your chart's appearance.

The Chart data region presents a series of points in different ways depending upon the chart type you choose. Some chart types display multiple series of data points in a single chart. Add more information to your chart by configuring data points, axes, titles, and labels. You can modify all of these elements in the Properties Window.

When you first drop a Chart data region onto a report, the Chart Wizard appears, and you can set up your chart type, appearance, series, titles, axes, and legend on the pages of the wizard. You can specify a data source on the Series page.

Important Properties

Property	Description
BlackAndWhiteMode	Gets or sets a value indicating whether the chart is drawn in black and white using hatch patterns and line dashing to designate colors.
AutoRefresh	Gets or sets a value indicating whether the chart is automatically refreshed (redrawn) after every property change.
Backdrop	Gets or sets the chart's background style.
ChartAreas	Opens the ChartArea Collection Editor where you can set properties such as axes and wall ranges, and you can add more chart areas.
ChartBorder	Gets or sets the chart's border style.
ColorPalette	Gets or sets the chart's color palette.
DataSource	Gets or sets the data source for the chart.
GridLayout	Gets or sets the layout of the chart's areas in columns and rows.
Legends	Opens the Legend Collection Editor where you can set up the chart's legends.
Series	Opens the Series Collection Editor where you can set up the series collection for the chart.
Titles	Opens the Titles Collection Editor where you can set up titles in the header and footer of the chart.
UIOptions	Gets or sets user interface features for the chart. Choose from None, ContextCustomize, UseCustomTooltips, or ForceHitTesting.
Culture	Gets or sets the chart's culture used for value output formatting.
ImageType	Sets or returns the image generated by the chart. Choose from Metafile or PNG.

Chart Commands and Dialogs

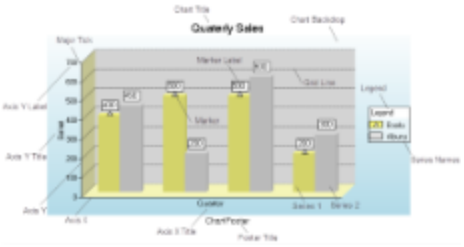
With the control selected on the report, in the Commands section at the bottom of the Properties window, you can click any of the commands to open a dialog. Commands in this section include:

- **Clear Chart** clears all of the property settings from the chart so that you can begin with a clean slate. You are given an opportunity to cancel this action.
- **Load** allows you to load a saved XML file containing a chart that you created using the Chart data region.
- **Save As** allows you to save the current chart to an XML file that you can load into a chart on any section report.
- **Customize** opens the main Chart Designer dialog where you can access Chart Areas, Titles, Series, Legends, and Appearance tabs. This dialog has access to more of the customizable areas than the wizard, but all of the

properties in this dialog are also available in the Properties window.
- **Wizard** reopens the Chart Wizard that appears by default when you first drop a Chart data region onto a report.
- **Data Source** opens the Chart Data Source dialog where you can build a connection string and create a query.

Chart Elements

The Chart elements help you to easily analyze the visual information and interpret numerical and relational data. The following image illustrates the elements that make up the Chart data region.



Axis Label

A label along an axis that lets you label the units being shown.

Axis Title

The axis title allows you to provide a title for the information being shown on the axis.

Chart Backdrop

The chart backdrop is the background for the whole chart that is created. You can create your own backdrop using the different styles and colors available or you can use an image as a backdrop for your chart.

Chart Title

The chart title serves as the title for the chart.

Footer Title

The footer title allows you to add a secondary title for the chart along the bottom.

Grid Line

Grid lines can occur on horizontal and vertical axes and normally correlate to the major or minor tick marks for the axes.

Legend

The legend serves as a key to the specific colors or patterns being used to show series values in the chart.

Marker

The marker is used to annotate a specific plotted point in a data series.

Marker Label

The marker label allows you to display the value of a specific plotted point in a data series.

Major Tick

Major tick marks can occur on horizontal and vertical axes and normally correlate to the major gridlines for the axes.

Minor Tick

Minor tick marks can occur on horizontal and vertical axes and normally correlate to the minor gridlines for the axes.

Series

The series is a related group of data values that are plotted on the chart. Each plotted point is a data point that reflects the specific values charted. Most charts, such as the above bar chart, can contain more than one series, while others, such as a pie chart, can contain only one.

Wall Backdrop

The wall is the back section of the chart on which data is plotted.

Chart Wizard

The Chart data region features a Chart Wizard which takes you through the basic steps of creating a chart. The **Chart Wizard** automatically appears when you first add a chart control to a report. If you prefer not to have the wizard appear automatically, clear the **Auto Run Wizard** checkbox at the bottom of the wizard.

The Chart Wizard has the following pages:

Chart Type

In the Chart Wizard that appears, the Chart Type page displays all available 2D and 3D chart types, along with a preview of the selected chart to the right. You can select the type of chart that you want to create and change the axes by selecting the **Swap Axes** checkbox. If you are using a 3D chart, you can also change the **projection** and **light** settings.

Appearance

The Appearance page has two tabs. The **Palette** tab allows you to select a color scheme. The **Appearance** tab allows you to select individual elements in the chart preview, such as its title, footer, legend, legend title, backdrop, and the chart itself and select appearance settings for them.

Series

The Series page has two tabs. The **Series Settings** tab allows you to set the data source for the chart and bind data fields to X and Y values for each series in the chart, and to add and remove chart series. You can even set different chart types for each series. The **Data Points** tab allows you to set static data values when you choose not to bind the X and Y values to data fields.

Title

The Title Page helps you to set properties for the header and footer titles. You can change the title text, font size and color, border settings, background color and visibility.

Axes

The Axes page has two tabs, one for **Axis X** and the other for **Axis Y**. On these tabs, you can enter titles for the axes and set the font size and other font properties. This page also allows you to add and format labels, add tick marks and grid lines, and select whether to show the axis inside or outside the chart area.

Legend

The Legend page allows you to set up the appearance of the legend. You can change its visibility, label appearance, header and footer text and appearance, layout, and location within the chart. You can also place the legend inside the chart by checking the **Legend inside** check box in the **Position** section.

Chart Types (Section Reports)

These topics introduce you to the different Chart Types you can create with the Chart control.

Area Chart
: Area2D, StackedArea, StackedArea100Pct, and Area3D

Bar Chart
: Bar2D and Bar3D

Line Chart
: Bezier, Line, LineXY and Line3D

Pie and Doughnut Charts
: Doughnut/Pie and Doughnut3D/Pie

Financial Chart
: Candle, HiLo, Renko, Point and Figure, Kagi, Stock, StockOpenClose, and Three Line Break

Point and Bubble Charts
: Bubble, BubbleXY, Scatter, and PlotXY

Area Chart

The **Area Chart** displays quantitative data and is based on the Line chart. In Area charts, the space between axis and line are commonly emphasized with colors, textures and hatchings.

2D Area Charts
This section describes 2D charts that fall under the Area Chart category.
3D Area Charts
This section describes 3D charts that fall under the Area Chart category.

2D Area Charts

Given below is the list of 2D charts that fall under the Area Chart category:

Area Chart

An area chart is used to compare trends over a period of time or across categories.

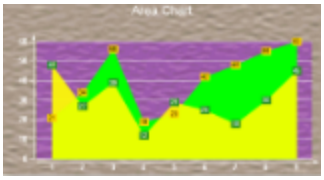


Chart Information	
ChartType	Area2D
Number of Y values per data point	1
Number of Series	1 or more
Marker Support	Series or Data Point
Custom Properties	None

Stacked Area Chart

A stacked area chart is an area chart with two or more data series stacked one on top of the other. Use this chart to show how each value contributes to a total.

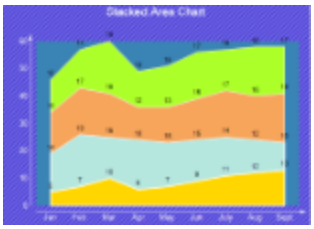


Chart Information	

ChartType	Area2D
Number of Y values per data point	1
Number of Series	1 or more
Marker Support	Series or Data Point
Custom Properties	None

Stacked Area 100% Chart

A stacked area chart (100%) is an 100% area chart with two or more data series stacked one on top of the other. Use this chart to show how each value contributes to a total.

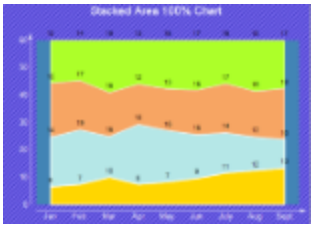


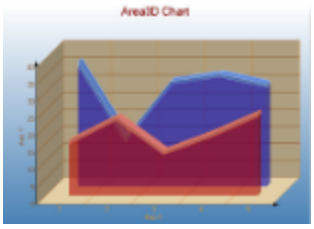
Chart Information	
ChartType	Area2D
Number of Y values per data point	1
Number of Series	1 or more
Marker Support	Series or Data Point
Custom Properties	None

3D Area Charts

Given below is the list of 3D charts that fall under the Area Chart category:

Area Chart

Use a 3D area chart to compare trends in two or more data series over a period of time or in specific categories, so that data can be viewed side by side.



⚠ **Note:** To view a chart in 3D, in the **ChartAreas** property open the **ChartArea Collection Editor** and set the

> **ProjectionType** property to Orthogonal.

Chart Information	
ChartType	Area3D
Number of Y values per data point	1
Number of Series	1 or more
Marker Support	Series or Data Point
Custom Properties	The **LineBackdrop** property gets or sets the backdrop information for the 3D line. The **Thickness** property gets or sets the thickness of the 3D line. The **Width** property gets or sets the width of the 3D line.

Below is an example of how to set the custom chart properties at run time for a 3D area chart as shown for the first series in the image above.

Visual Basic

```
Me.ChartControl1.Series(0).Properties("LineBackdrop") = New
GrapeCity.ActiveReports.Chart.Graphics.Backdrop(Color.Red, CType(150, Byte))
Me.ChartControl1.Series(0).Properties("Thickness") = 5.0F
Me.ChartControl1.Series(0).Properties("Width") = 30.0F
```

C#

```
this.ChartControl1.Series[0].Properties["LineBackdrop"] = new
GrapeCity.ActiveReports.Chart.Graphics.Backdrop(System.Drawing.Color.Red, ((System.Byte)
(150)));
this.ChartControl1.Series[0].Properties["Thickness"] = 5f;
this.ChartControl1.Series[0].Properties["Width"] = 30f;
```

Stacked Area Chart

3D stacked area chart displays stacked area chart in 3D.

Chart Information	
ChartType	Area3D
Number of Y values per data point	1
Number of Series	1 or more
Marker Support	Series or Data Point
Custom Properties	The **Width** property gets or sets the width of the 3D stacked area.

Stacked Area 100%

3D stacked area chart (100%) displays stacked area chart in 3D (100%).

Chart Information	
ChartType	Area3D
Number of Y values per data point	1
Number of Series	1 or more
Marker Support	3
Custom Properties	The **Width** property gets or sets the width of the 3D stacked area.

Bar Chart

The **Bar Chart** is a chart with rectangular bars where the lengths of bars are proportional to the values they represent. The bars can be plotted vertically or horizontally.

2D Bar Charts
: This section describes 2D charts that fall under the Bar Chart category.

3D Bar Charts
: This section describes 3D charts that fall under the Bar Chart category.

2D Bar Charts

Given below is the list of 2D charts that falls under the Bar Chart category.

Bar Chart

In a Bar Chart, values are represented by the height of the bar shaped marker as measured by the y-axis. Category labels are displayed on the x-axis. Use a bar chart to compare values of items across categories.

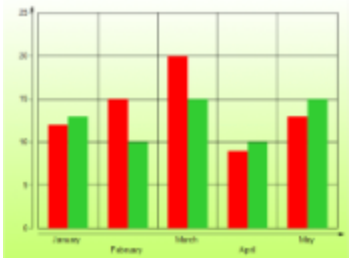


Chart Information	
ChartType	Bar2D
Number of Y values per data point	1
Number of series	1 or more
Marker support	Series or Data Point

Custom Properties	The **Gap** property gets or sets the space between the bars of each X axis value.

Below is an example of how to set the custom chart properties at run time for a bar chart.

Visual Basic

```
Me.ChartControl1.Series(0).Properties("Gap") = 50.0F
```

C#

```
this.ChartControl1.Series[0].Properties["Gap"] = 50f;
```

Gantt Chart

The Gantt chart is a project management tool used to chart the progress of individual project tasks. The chart compares project task completion to the task schedule.

⚠ **Caution**: In a Gantt chart, the X and Y axes are reversed. AxisX is vertical and AxisY is horizontal.

Chart Information	
ChartType	Bar2D
Number of Y values per data point	2
Number of Series	1 or more
Marker Support	Series or Data Point
Custom Properties	The **Gap** property gets or sets the space between the bars of each X axis value.

Below is an example of how to set the custom chart properties at run time for a Gantt chart.

Visual Basic

```
Me.ChartControl1.Series(0).Properties("Gap") = 50.0F
```

C#

```
this.ChartControl1.Series[0].Properties["Gap"] = 50f;
```

Horizontal Bar Chart

In a Horizontal Bar Chart, both the axes are swapped and therefore the bars appears horizontally. Although, values are represented by the height of the bar shaped marker as measured by the y-axis and the Category labels are displayed on the x-axis. Use a horizontal bar chart to compare values of items across categories.

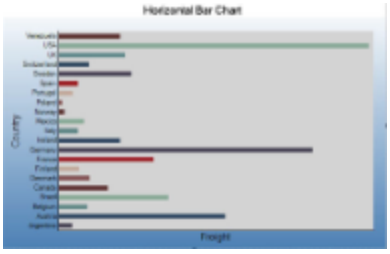


Chart Information	
ChartType	Bar2D
Number of Y values per data point	1
Number of Series	1 or more
Marker Support	Series or Data Point
Custom Properties	The **Gap** property gets or sets the space between the bars of each X axis value.

Below is an example of how to set the custom chart properties at run time for a horizontal bar chart.

Visual Basic
`Me.ChartControl1.Series(0).Properties("Gap") = 65.0F`
C#
`this.ChartControl1.Series[0].Properties["Gap"] = 65f;`

Stacked Bar Chart

A stacked bar chart is a bar chart with two or more data series stacked one on top of the other. Use this chart to show how each value contributes to a total.

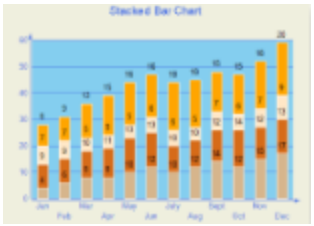


Chart Information	
ChartType	Bar2D
Number of Y values per data point	1
Number of Series	1 or more
Marker Support	Series or Data Point
Custom Properties	The **Gap** property gets or sets the space between the bars of each X axis value.

Below is an example of how to set the custom chart properties at run time for a StackedBar chart.

Visual Basic
`Me.ChartControl1.Series(0).Properties("Gap") = 100.0F`
C#
`this.ChartControl1.Series[0].Properties["Gap"] = 100f;`

Stacked Bar Chart 100%

A StackedBAR110Pct chart is a bar chart with two or more data series stacked one on top of the other to sum up to 100%. Use this chart to show how each value contributes to a total with the relative size of each series representing its contribution to the total.

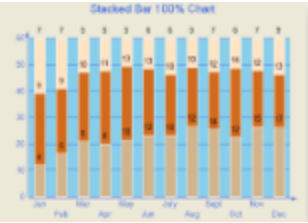


Chart Information	
ChartType	Bar2D
Number of Y values per data point	1
Number of Series	1 or more
Marker Support	Series or Data Point
Custom Properties	The **Gap** property gets or sets the space between the bars of each X axis value

Below is an example of how to set the custom chart properties at run time for a StackedBAR110Pct chart.

Visual Basic
`Me.ChartControl1.Series(0).Properties("Gap") = 100.0F`
C#
`this.ChartControl1.Series[0].Properties["Gap"] = 100f;`

3D Bar Charts

Given below is the list of 3D charts that fall under the Bar Chart category.

> ⚠ **Caution**: To view a chart in 3D, open the **ChartArea Collection Editor** in the **ChartAreas** property and set the **ProjectionType** property to Orthogonal.

Bar Chart

In a Bar Chart, values are represented by the height of the bar shaped marker as measured by the y-axis. Category labels are displayed on the x-axis. Use a 3D bar chart to compare values of items across categories, allowing the data to be viewed in a

convenient 3D format.

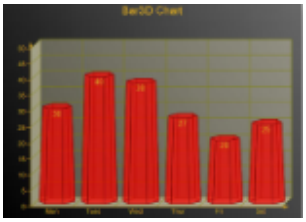


Chart Information	
ChartType	Bar3D
Number of Y values per data point	1
Number of Series	1 or more
Marker Support	Series or Data Point
Custom Properties	The **BarTopPercent** property gets or sets the percentage of the top of the bar that is shown for Cone or Custom BarTypes. The **BarType** property gets or sets the type of bars that is displayed.Use BarType enumeration value. The **Gap** property gets or sets the space between the bars of each X axis value. The **RotationAngle** propertygets or sets the starting horizontal angle for custom 3D bar shapes. Can only be used with the Custom BarType. The **VertexNumber** property gets or sets the number of vertices for the data point, used to create custom 3D bar shapes. Can only be used with the Custom BarType. Bars must contain 3 or more vertices.

Gantt Chart

The 3D gantt chart displays a gantt chart in 3D.

> ⚠ **Caution**: In a 3D Gantt chart the X and Y axes are reversed. AxisX is vertical and AxisY is horizontal.

Chart Information	
ChartType	Bar3D
Number of Y values per data point	2
Number of Series	1 or more
Marker Support	Series or Data Point
Custom Properties	The **BarTopPercent** property gets or sets the percentage of the top of the bar that is shown for Cone or Custom BarTypes. The **BarType** property gets or sets the type of bars that are displayed. Use BarType enumeration value. The **Gap** property gets or sets the space between the bars of each X axis value. The **VertexNumber** property gets or sets the number of vertices for the data point, used to create custom 3D bar shapes. Can only be used with the Custom BarType. Bars must contain 3 or more vertices.

Horizontal Bar Chart

In a Horizontal Bar Chart, both the axes are swapped and therefore the bars appears horizontally. Although, values are represented by the height of the bar shaped marker as measured by the y-axis and the Category labels are displayed on the x-axis. Use a horizontal 3D bar chart to compare values of items across categories, allowing the data to be viewed in a convenient 3D format.

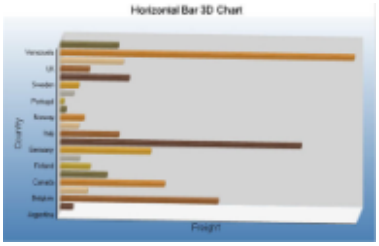


Chart Information	
ChartType	Bar3D
Number of Y values per data point	1
Number of Series	1 or more
Marker Support	Series or Data Point
Custom Properties	The **BarTopPercent** property gets or sets the percentage of the top of the bar that is shown for Cone or Custom BarTypes. The **BarType** propertygets or sets the type of bars that is displayed. Use BarType enumeration value. The **Gap** property gets or sets the space between the bars of each X axis value. The **RotationAngle** property gets or sets the starting horizontal angle for custom 3D bar shapes. Can only be used with the Custom BarType. The **VertexNumber** property gets or sets the number of vertices for the data point, used to create custom 3D bar shapes. Can only be used with the Custom BarType. Bars must contain 3 or more vertices.

Below is an example of how to set the custom chart properties at run time for a horizontal 3D bar chart as shown above.

Visual Basic

```
Me.ChartControl1.Series(0).Properties("BarTopPercent") = 80.0F
Me.ChartControl1.Series(0).Properties("BarType") =
GrapeCity.ActiveReports.Chart.BarType.Custom
Me.ChartControl1.Series(0).Properties("Gap") = 65.0F
Me.ChartControl1.Series(0).Properties("PointBarDepth") = 100.0F
Me.ChartControl1.Series(0).Properties("RotationAngle") = 0.0F
Me.ChartControl1.Series(0).Properties("VertexNumber") = 6
```

C#

```
this.ChartControl1.Series[0].Properties["BarTopPercent"] = 80f;
this.ChartControl1.Series[0].Properties["BarType"] =
GrapeCity.ActiveReports.Chart.BarType.Custom;
```

```
this.ChartControl1.Series[0].Properties["Gap"] = 65f;
this.ChartControl1.Series[0].Properties["PointBarDepth"] = 100.0f;
this.ChartControl1.Series[0].Properties["RotationAngle"] = 0f;
this.ChartControl1.Series[0].Properties["VertexNumber"] = 6;
```

Stacked Bar Chart

Use a 3D bar graph to compare values of items across categories, allowing the data to be viewed conveniently in a 3D format. A stacked bar graph is a bar graph with two or more data series stacked on top of each other. Use this graph to show how each value contributes to a total.

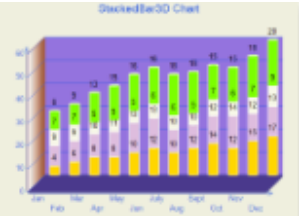


Chart Information	
ChartType	Bar3D
Number of Y values per data point	1
Number of Series	1 or more
Marker Support	Series or Data Point
Custom Properties	The **BarTopPercent** property gets or sets the percentage of the top of the bar that is shown for Cone or Custom BarTypes. The **BarType** property gets or sets the type of bars that are displayed. Use BarType enumeration value. The **Gap** property gets or sets the space between the bars of each X axis value. The **VertexNumber** property gets or sets the number of vertices for the data point, used to create custom 3D bar shapes. Can only be used with the Custom BarType. Bars must contain 3 or more vertices.

Below is an example of how to set the custom chart properties at run time for a StackedBar3D chart.

Visual Basic

```
Me.ChartControl1.Series(0).Properties("BarTopPercent") = 80.0F
Me.ChartControl1.Series(0).Properties("BarType") =
GrapeCity.ActiveReports.Chart.BarType.Custom
Me.ChartControl1.Series(0).Properties("Gap") = 65.0F
Me.ChartControl1.Series(0).Properties("VertexNumber") = 6
```

C#

```
this.ChartControl1.Series[0].Properties["BarTopPercent"] = 100f;
this.ChartControl1.Series[0].Properties["BarType"] =
GrapeCity.ActiveReports.Chart.BarType.Custom;
this.ChartControl1.Series[0].Properties["Gap"] = 65f;
this.ChartControl1.Series[0].Properties["VertexNumber"] = 6
```

Stacked Bar Chart 100%

A Stacked Bar 3D 100% chart is a bar chart with two or more data series in 3D stacked one on top of the other to sum up to 100%. Use this chart to show how each value contributes to a total with the relative size of each series representing its contribution to the total.

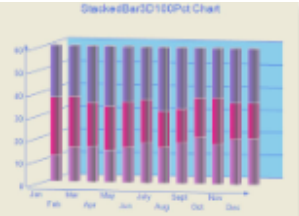


Chart Information	
ChartType	Bar3D
Number of Y values per data point	1
Number of Series	1 or more
Marker Support	Series or Data Point
Custom Properties	The **BarTopPercent** property gets or sets the percentage of the top of the bar that is shown for Cone or Custom BarTypes. The **BarType** property gets or sets the type of bars that are displayed. Use BarType enumeration value. The **Gap** property gets or sets the space between the bars of each X axis value. The **VertexNumber** property gets or sets the number of vertices for the data point, used to create custom 3D bar shapes. Can only be used with the Custom BarType. Bars must contain 3 or more vertices.

Below is an example of how to set the custom chart properties at run time for a StackedBar3D100Pct chart.

Visual Basic

```
Me.ChartControl1.Series(0).Properties("BarTopPercent") = 80.0F
Me.ChartControl1.Series(0).Properties("BarType") =
GrapeCity.ActiveReports.Chart.BarType.Custom
Me.ChartControl1.Series(0).Properties("Gap") = 65.0F
Me.ChartControl1.Series(0).Properties("VertexNumber") = 6
```

C#

```
this.ChartControl1.Series[0].Properties["BarTopPercent"] = 100f;
this.ChartControl1.Series[0].Properties["BarType"] =
GrapeCity.ActiveReports.Chart.BarType.Custom;
this.ChartControl1.Series[0].Properties["Gap"] = 65f;
this.ChartControl1.Series[0].Properties["VertexNumber"] = 6
```

Bar/Cylinder Chart

It is almost similar to a bar chart and values are represented by the height of the bar shaped marker as measured by the y-axis. Category labels are displayed on the x-axis. The only difference is that in a Bar/Cylinder Chart the data is represented

through cylindrical shaped markers.

Chart Information	
ChartType	Bar3D
Number of Y values per data point	1
Number of Series	1 or more
Marker Support	Series or Data Point
Custom Properties	The **BarTopPercent** property gets or sets the percentage of the top of the bar that is shown for Cone or Custom BarTypes. The **BarType** property gets or sets the type of bars that is displayed. Use BarType enumeration value. The **Gap** property gets or sets the space between the bars of each X axis value. The **RotationAngle** property gets or sets the starting horizontal angle for custom 3D bar shapes. Can only be used with the Custom BarType. The **VertexNumber** property gets or sets the number of vertices for the data point, used to create custom 3D bar shapes. Can only be used with the Custom BarType. Bars must contain 3 or more vertices.

Bar/Pyramid Chart

In a Bar/Pyramid Chart the data is represented through pyramid shaped bars and values are represented by the height of the bars as measured by the y-axis. Category labels are displayed on the x-axis.

Chart Information	
ChartType	Bar3D
Number of Y values per data point	1
Number of Series	1 or more
Marker Support	Series or Data Point
Custom Properties	The **BarTopPercent** property gets or sets the percentage of the top of the bar that is shown for Cone or Custom BarTypes. The **BarType** property gets or sets the type of bars that is displayed. Use BarType enumeration value. The **Gap** property gets or sets the space between the bars of each X axis value. The **RotationAngle** property gets or sets the starting horizontal angle for custom 3D bar shapes. Can only be used with the Custom BarType. The **VertexNumber** property gets or sets the number of vertices for the data point, used to create custom 3D bar shapes. Can only be used with the Custom BarType. Bars must contain 3 or more vertices.

Clustered Bar chart

Use a 3D clustered bar chart to compare values of items across categories, allowing the data to be viewed in a convenient 3D

format.

Chart Information	
ChartType	Bar3D
Number of Y values per data point	1
Number of Series	1 or more
Marker Support	Series or Data Point
Custom Properties	The **BarTopPercent** property gets or sets the percentage of the top of the bar that is shown for Cone or Custom BarTypes. The **BarType** property gets or sets the type of bars that are displayed. Use BarType enumeration value. The **Gap** property gets or sets the space between the bars of each X axis value. The **RotationAngle** property gets or sets the starting horizontal angle for custom 3D bar shapes. Can only be used with the Custom BarType. The **VertexNumber** property gets or sets the number of vertices for the data point, used to create custom 3D bar shapes. Can only be used with the Custom BarType. Bars must contain 3 or more vertices.

Below is an example of how to set the custom chart properties at run time for a 3D clustered bar chart as shown above.

Visual Basic

```
' set the custom properties for series 1.
Me.ChartControl1.Series(0).Properties("BarTopPercent") = 50.0F
Me.ChartControl1.Series(0).Properties("BarType") =
GrapeCity.ActiveReports.Chart.BarType.Custom
Me.ChartControl1.Series(0).Properties("Gap") = 300.0F
Me.ChartControl1.Series(0).Properties("RotationAngle") = 0.0F
Me.ChartControl1.Series(0).Properties("VertexNumber") = 6

' set the custom properties for series 2.
Me.ChartControl1.Series(1).Properties("BarTopPercent") = 20.0F
Me.ChartControl1.Series(1).Properties("BarType") =
GrapeCity.ActiveReports.Chart.BarType.Custom
Me.ChartControl1.Series(1).Properties("Gap") = 300.0F
Me.ChartControl1.Series(1).Properties("RotationAngle") = 90.0F
Me.ChartControl1.Series(1).Properties("VertexNumber") = 3
```

C#

```
// set the custom properties for series 1.
this.ChartControl1.Series[0].Properties["BarTopPercent"] = 50f;
```

```
this.ChartControl1.Series[0].Properties["BarType"] =
GrapeCity.ActiveReports.Chart.BarType.Custom;
this.ChartControl1.Series[0].Properties["RotationAngle"] = 0f;
this.ChartControl1.Series[0].Properties["VertexNumber"] = 6;

// set the custom properties for series 2.
this.ChartControl1.Series[1].Properties["BarTopPercent"] = 20f;
this.ChartControl1.Series[1].Properties["BarType"] =
GrapeCity.ActiveReports.Chart.BarType.Custom;
this.ChartControl1.Series[1].Properties["Gap"] = 300f;
this.ChartControl1.Series[1].Properties["RotationAngle"] = 90f;
this.ChartControl1.Series[1].Properties["VertexNumber"] = 3;
```

Line Chart

The **Line Chart** is a type of chart that displays information as a series of data points, connected by straight line segments.

2D Line Charts
: This section describes 2D charts that fall under the Line Chart category.

3D Line Charts
: This section describes 3D charts that fall under the Line Chart category.

2D Line Charts

Given below is the list of 2D charts that fall under the Line Chart category.

Bezier Chart

Use a Bezier or spline chart to compare trends over a period of time or across categories. It is a line chart that plots curves through the data points in a series.

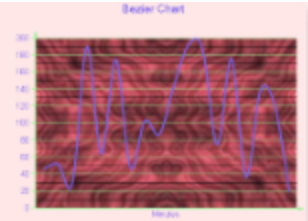


Chart Information	
ChartType	Line2D
Number of Y values per data point	1
Number of Series	1 or more
Marker Support	Series or Data Point
Custom Properties	The **Line** property gets or sets the line element. Used to set color, thickness and shape of a line.

	The **Tension** property gets or sets the tension of the curved lines.

Bezier XY Chart

A Bezier XY chart connects DataPoints on X and Y with curved lines.

Chart Information	
ChartType	Line2D
Number of Y values per data point	1
Number of Series	1 or more
Marker Support	Series or Data Point
Custom Properties	The **Line** property gets or sets the line element. Used to set color, thickness and shape of a line. The **Tension** property gets or sets the tension of the curved lines.

Line Chart

Use a 2D line chart to compare trends over a period of time or in certain categories in a 2D format.

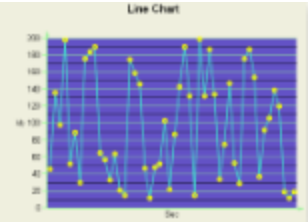


Chart Information	
ChartType	Line2D
Number of Y values per data point	1
Number of Series	1 or more
Marker Support	Series or Data Point
Custom Properties	The **Line** property gets or sets the line element. Used to set color, thickness and shape of a line. The **LineJoin** property sets the type of join to draw when two lines connect.

Line XY Chart

A line XY chart plots points on the X and Y axes as one series and uses a line to connect points to each other.

Chart Information	
ChartType	Line2D
Number of Y values per data point	1
Number of Series	1 or more
Marker Support	Series or Data Point
Custom Properties	The **Line** property gets or sets line elements. Used for setting color, thickness and shape of a line. The **LineJoin** property gets or sets the type of join to draw when two lines connect.

3D Line Charts

Given below is the list of 3D charts that fall under the Line Chart category.

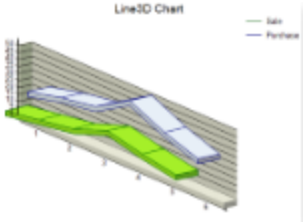
Bezier Chart

Render a Bezier or Spline chart in 3D format.

Chart Information	
ChartType	Line3D
Number of Y values per data point	1
Number of Series	1 or more
Marker Support	Series or Data Point
Custom Properties	The **LineBackdrop** property gets or sets the backdrop information for the 3D curved line. The **Tension** property gets or sets the tension of the curved lines. The **Width** property gets or sets the width of the 3D curved line.

Line Chart

Use a 3D line chart to compare trends over a period of time or in certain categories in a 3D format.



> ⚠ **Caution**: To view a chart in 3D, open the **ChartArea Collection Editor** in the **ChartAreas** property and set the **ProjectionType** property to Orthogonal.

Chart Information	
ChartType	Line3D
Number of Y values per data point	1
Number of Series	1or more
Marker Support	Series or Data Point
Custom Properties	The **LineBackdrop** property gets or sets the backdrop information for the 3D line. The **Thickness** property gets or sets the thickness of the 3D line. The **Width** property gets or sets the width of the 3D line.

Below is an example of how to set the custom chart properties at run time for a horizontal 3D bar chart as shown above.

Visual Basic

```
Me.ChartControl1.Series(0).Properties("LineBackdrop") = New Backdrop(Color.GreenYellow)
Me.ChartControl1.Series(0).Properties("Thickness") = 8.0F
Me.ChartControl1.Series(0).Properties("Width") = 40.0F
```

C#

```
this.chartControl1.Series[0].Properties["LineBackdrop"] = new
Backdrop(Color.GreenYellow);
this.chartControl1.Series[0].Properties["Thickness"] = 8f;
this.chartControl1.Series[0].Properties["Width"] = 40f;
```

Pie and Doughnut Charts

A **Pie Chart** is a circular chart divided into sectors to illustrate proportion.

A **Doughnut Chart** is functionally identical to a **Pie Charts.** It also has single-series and multi-series versions, with the only difference that it has a hole in the middle.

2D Pie/Doughnut Charts
: This section describes 2D charts that fall under the Pie/Doughnut Chart category.

3D Pie/Doughnut Charts
: This section describes 3D charts that fall under the Pie/Doughnut Chart category.

2D Pie/Doughnut Charts

Given below is the list of 2D charts that fall under the Pie/Doughnut Chart category.

Doughnut Chart

A doughnut chart shows how the percentage of each data item contributes to the total.

⚠ In order to show each section of the pie in a different color, set the **Background** property for each data point.

Chart Information	
ChartType	Pie/Doughnut 2D
Number of Y values per data point	1
Number of Series	1 or more
Marker Support	Series or Data Point
Custom Properties	The **Clockwise** property gets or sets a value indicating whether to display the data in clockwise order. The **ExplodeFactor** property gets or sets the amount of separation between data point values. The **HoleSize** property gets or sets the inner radius of the chart. The **OutsideLabels** property gets or sets a value indicating whether the data point labels appear outside the chart. The **StartAngle** property gets or sets the horizontal start angle for the series.

Below is an example of how to set custom chart properties at run time for a doughnut chart.

Visual Basic

```
Me.ChartControl1.Series(0).Properties("ExplodeFactor") = 0.0F
Me.ChartControl1.Series(0).Properties("HoleSize") = 0.25F
Me.ChartControl1.Series(0).Properties("OutsideLabels") = False
Me.ChartControl1.Series(0).Properties("Radius") = 2.0F
Me.ChartControl1.Series(0).Properties("StartAngle") = 0.0F
```

C#

```
this.ChartControl1.Series[0].Properties["ExplodeFactor"] = 0f;
this.ChartControl1.Series[0].Properties["HoleSize"] = 0.25f;
this.ChartControl1.Series[0].Properties["OutsideLabels"] = false;
this.ChartControl1.Series[0].Properties["Radius"] = 2.0f;
this.ChartControl1.Series[0].Properties["StartAngle"] = 0f;
```

Funnel Chart

A funnel chart shows how the percentage of each data item contributes as a whole.

Chart Information	
ChartType	Pie/Doughnut 2D
Number of Y values per data points	1
Number of Series	1 or more
Marker Support	Series or Data Point
Custom Properties	The **CalloutLine** property gets or sets the style for a line connecting the marker label to its corresponding funnel section. The default value is a black one-point line. The **FunnelStyle** property gets or sets the Y value for the series points to the width or height of the funnel. The default value is YIsHeight. The **MinPointHeight** property gets or sets the minimum height allowed for a data point in the funnel chart. The height is measured in relative coordinates. The **NeckHeight** property gets or sets the neck height for the funnel chart. This property can only be used with the FunnelStyle property set to YIsHeight. The default value is 5. The **NeckWidth** property gets or sets the neck width for the funnel chart. This property can only be used with the FunnelStyle property set to YIsHeight. The default value is 5. The **OutsideLabels** property gets or sets a value indicating whether the labels are placed outside of the funnel chart. The default value is True. The **OutsideLabelsPlacement** property gets or sets a value indicating whether the data point labels appear on the left or right side of the funnel. This property can only be used with the OutsideLabels property set to True. The **PointGapPct** property gets or sets the amount of space between the data points of the funnel chart. The PointGapPct is measured in relative coordinates. The default value is 0, and valid values range from 0 to 100.

Pyramid Chart

A Pyramid chart shows how the percentage of each data item contributes as a whole.

Chart Information	
ChartType	Pie/Doughnut 2D
Number of Y values per data point	1
Number of	1 or more

Series	
Marker Support	Series or Data Points
Custom Properties	The **CalloutLine** property gets or sets the style for a line connecting the marker label to its corresponding pyramid section. The default value is a black one-point line. The **MinPointHeight** property gets or sets the minimum height allowed for a data point in the pyramid chart. The height is measured in relative coordinates. The **OutsideLabels** property gets or sets a value indicating whether the labels are placed outside of the pyramid chart. The default value is True. The **OutsideLabelsPlacement** property gets or sets a value indicating whether the data point labels appear on the left or right side of the pyramid. This property can only be used with the OutsideLabels property set to True. The **PointGapPct** property gets or sets the amount of space between the data points of the pyramid chart. The PointGapPct is measured in relative coordinates. The default value is 0, and valid values range from 0 to 100.

3D Pie/Doughnut Charts

Given below is the list of 3D charts that fall under the Pie/Doughnut Chart category.

> ⚠ **Caution**: To view a chart in 3D, open the **ChartArea Collection Editor** in the **ChartAreas** property and set the **ProjectionType** property to Orthogonal.

Doughnut Chart

A 3D doughnut chart shows how the percentage of each data item contributes to a total percentage, allowing the data to be viewed in a 3D format.

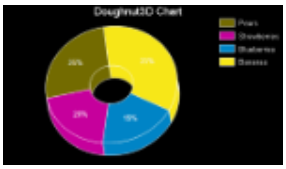


Chart Information	
ChartType	Pie/Doughnut 3D
Number of Y values per data point	1
Number of Series	1 or more
Marker Support	Series or Data Point
Custom Properties	The **Clockwise** property gets or sets a value indicating whether to display the data in clockwise order. The **ExplodeFactor** property gets or sets the amount of separation between data point values. The value must be less than or equal to 1. To explode one section of the doughnut chart, set ExplodeFactor to the data point instead of the series. The **HoleSize** property gets or sets the inner radius of the chart. If set to 0, the chart looks like a pie chart. The value must be less than or equal to 1.

	The **OutsideLabels** property gets or sets a value indicating whether the data point labels appear outside the chart. The **Radius** property gets or sets the size of the doughnut within the chart area. The **StartAngle** property gets or sets the horizontal start angle for the series data points.

Below is an example of how to set the custom chart properties at run time for a 3D doughnut chart as shown in the image above.

To write the code in Visual Basic.NET

Visual Basic

```
Me.ChartControl1.Series(0).Properties("ExplodeFactor") = 0.0F
Me.ChartControl1.Series(0).Properties("HoleSize") = 0.33F
Me.ChartControl1.Series(0).Properties("OutsideLabels") = False
Me.ChartControl1.Series(0).Properties("Radius") = 2.0F
Me.ChartControl1.Series(0).Properties("StartAngle") = 50.0F
```

To write the code in C#

C#

```
this.chartControl1.Series[0].Properties["ExplodeFactor"] = 0f;
this.chartControl1.Series[0].Properties["HoleSize"] = .33f;
this.chartControl1.Series[0].Properties["OutsideLabels"] = false;
this.chartControl1.Series[0].Properties["StartAngle"] = 50f;
```

Funnel Chart

A 3D funnel chart shows how the percentage of each data item contributes to the whole, allowing the data to be viewed in a 3D format.

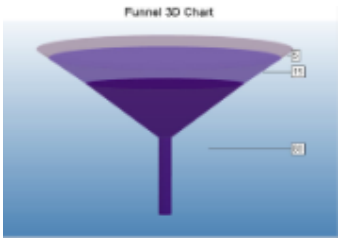


Chart Information	
ChartType	Pie/Doughnut 3D
Number of Y values per data points	1
Number of Series	1 or more
Marker Support	Series or Data Point
	The **BaseStyle** property gets or sets a circular or square base drawing style for the 3D funnel chart. The **CalloutLine** property gets or sets the style for a line connecting the marker label to its corresponding funnel section. The default value is a black one-point line. The **FunnelStyle** property gets or sets the Y value for the series points to the width or height of the funnel. The

Custom Properties	default value is YIsHeight. The **MinPointHeight** property gets or sets the minimum height allowed for a data point in the funnel chart. The height is measured in relative coordinates. The **NeckHeight** property gets or sets the neck height for the funnel chart. This property can only be used with the FunnelStyle property set to YIsHeight. The default value is 5. The **NeckWidth** property gets or sets the neck width for the funnel chart. This property can only be used with the FunnelStyle property set to YIsHeight. The default value is 5. The **OutsideLabels** property gets or sets a value indicating whether the labels are placed outside of the funnel chart. The default value is True. The **OutsideLabelsPlacement** property gets or sets a value indicating whether the data point labels appear on the left or right side of the funnel. This property can only be used with the OutsideLabels property set to True. The **PointGapPct** property gets or sets the amount of space between the data points of the funnel chart. The PointGapPct is measured in relative coordinates. The default value is 0, and valid values range from 0 to 100. The **RotationAngle** property gets or sets the left-to-right rotation angle of the funnel. The valid values range from -180 to 180 degrees. This property is only effective with the Projection property set to Orthogonal and the BaseStyle property set to SquareBase.

Below is an example of how to set the custom chart properties at run time for a 3D funnel chart.

To write the code in Visual Basic.NET

Visual Basic

```
Imports GrapeCity.ActiveReports.Chart
Imports GrapeCity.ActiveReports.Chart.Graphics
```

Visual Basic

```
With Me.ChartControl1.Series(0)
    .Properties("BaseStyle") = BaseStyle.SquareBase
    .Properties("CalloutLine") = New Line(Color.Black, 2, LineStyle.Dot)
    .Properties("FunnelStyle") = FunnelStyle.YIsWidth
    .Properties("MinPointHeight") = 10.0F
    .Properties("NeckWidth") = 20.0F
    .Properties("NeckHeight") = 5.0F
    .Properties("OutsideLabels") = True
    .Properties("OutsideLabelsPlacement") = LabelsPlacement.Right
    .Properties("PointGapPct") = 3.0F
    .Properties("RotationAngle") = 3.0F
End With
```

To write the code in C#

C#

```
using GrapeCity.ActiveReports.Chart;
using GrapeCity.ActiveReports.Chart.Graphics;
```

C#

```
this.ChartControl1.Series[0].Properties["BaseStyle"] = BaseStyle.SquareBase;
this.ChartControl1.Series[0].Properties["CalloutLine"] = new Line(Color.Black, 2,
LineStyle.Dot);
```

```
this.ChartControl1.Series[0].Properties["FunnelStyle"] = FunnelStyle.YIsWidth;
this.ChartControl1.Series[0].Properties["MinPointHeight"] = 10f;
this.ChartControl1.Series[0].Properties["NeckWidth"] = 20f;
this.ChartControl1.Series[0].Properties["NeckHeight"] = 5f;
this.ChartControl1.Series[0].Properties["OutsideLabels"] = true;
this.ChartControl1.Series[0].Properties["OutsideLabelsPlacement"] = LabelsPlacement.Right;
this.ChartControl1.Series[0].Properties["PointGapPct"] = 3f;
this.ChartControl1.Series[0].Properties["RotationAngle"] = 3f;
```

Pyramid Chart

A 3D Pyramid chart shows how the percentage of each data item contributes to the whole, allowing the data to be viewed in a 3D format.

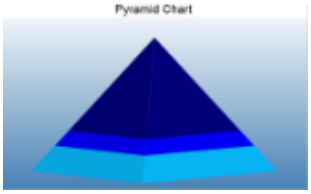


Chart Information	
ChartType	Pie/Doughnut 3D
Number of Y values per data point	1
Number of Series	1 or more
Marker Support	Series or Data Points
Custom Properties	The **BaseStyle** property gets or sets a circular or square base drawing style for the 3D pyramid chart. The **CalloutLine** property gets or sets the style for a line connecting the marker label to its corresponding pyramid section. The default value is a black one-point line. The **MinPointHeight** property gets or sets the minimum height allowed for a data point in the pyramid chart. The height is measured in relative coordinates. The **OutsideLabels** property gets or sets a value indicating whether the labels are placed outside of the pyramid chart. The default value is True. The **OutsideLabelsPlacement** property gets or sets a value indicating whether the data point labels appear on the left or right side of the pyramid. This property can only be used with the OutsideLabels property set to True. The **PointGapPct** property gets or sets the amount of space between the data points of the pyramid chart. The PointGapPct is measured in relative coordinates. The default value is 0, and valid values range from 0 to 100. The **RotationAngle** property gets or sets the left-to-right rotation angle of the pyramid. The valid values range from -180 to 180 degrees. This property is only effective with the Projection property set to Orthogonal and the BaseStyle property set to SquareBase.

Below is an example of how to set the custom chart properties at run time for a Pyramid chart.

To write the code in Visual Basic.NET

Visual Basic
`Imports GrapeCity.ActiveReports.Chart`

```
Imports GrapeCity.ActiveReports.Chart.Graphics
```

Visual Basic

```
With Me.ChartControl1.Series(0)
    .Properties("BaseStyle") = BaseStyle.SquareBase
    .Properties("MinPointHeight") = 10.0F
    .Properties("OutsideLabels") = True
    .Properties("OutsideLabelsPlacement") = LabelsPlacement.Right
    .Properties("PointGapPct") = 3.0F
    .Properties("RotationAngle") = 3.0F
End With
```

To write the code in C#

C#

```
using GrapeCity.ActiveReports.Chart;
using GrapeCity.ActiveReports.Chart.Graphics;
```

C#

```
this.ChartControl1.Series[0].Properties["BaseStyle"] = BaseStyle.SquareBase;
this.ChartControl1.Series[0].Properties["MinPointHeight"] = 10f;
this.ChartControl1.Series[0].Properties["OutsideLabels"] = true;
this.ChartControl1.Series[0].Properties["OutsideLabelsPlacement"] = LabelsPlacement.Right;
this.ChartControl1.Series[0].Properties["PointGapPct"] = 3f;
this.ChartControl1.Series[0].Properties["RotationAngle"] = 3f;
```

Pie Chart

This type of chart displays the contribution of each value to a total.

Chart Information	
ChartType	Pie/Doughnut 3D
Number of Y values per data point	1
Number of Series	1 or more
Marker Support	Series or Data Point
Custom Properties	

Ring Chart

This chart type uses rings (inner and outer) to represent data.

Chart Information

ChartType	Pie/Doughnut 3D
Number of Y values per data point	1
Number of Series	1 or more
Marker Support	Series or Data Point
Custom Properties	

Financial Chart

Financial charts are those which are specific to representing data related to financial activities. The Chart data region can draw a number of financial chart types:

- Candle, High Low, High Low Open Close
- Kagi, Renko
- Point and Figure, Three Line Break

2D Financial Charts
: This section describes 2D charts that fall under the Financial Chart category.

3D Financial Charts
: This section describes 3D charts that fall under the Financial Chart category.

2D Financial Charts

Given below is the list of 2D charts that fall under the Financial Chart category.

Candle Stick Chart

A candle chart displays stock information, using High, Low, Open and Close values. The size of the wick line is determined by the High and Low values, while the size of the bar is determined by the Open and Close values. The bar is displayed using different colors, depending on whether the price of the stock has gone up or down.

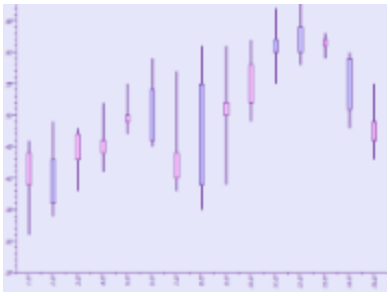


Chart Information	
ChartType	Financial2D
Number of Y values per data point	4 (The first value is the high figure, the second is the low figure, the third is the opening figure, and the fourth is the closing figure.)
Number of Series	1 or more

Marker Support	Series or Data Point. Marker labels use the first Y value as the default value.
Custom Properties	The **BodyDownswingBackdrop** property gets or sets the backdrop information used to fill the rectangle for data points in which the closing figure is lower than the opening figure. The **BodyUpswingBackdrop** property gets or sets the backdrop information used to fill the rectangle for data points in which the closing figure is higher than the opening figure. The **BodyWidth** property gets or sets the width of the rectangle used to show upswing or downswing. The **WickLine** property gets or sets the line information for the wick line.

Below is an example of how to set the custom chart properties at run time for a candle chart as shown in the image above.

To write the code in Visual Basic.NET

Visual Basic

```
Imports GrapeCity.ActiveReports.Chart.Graphics
```

Visual Basic

```
With Me.ChartControl1.Series(0)
.Properties("BodyDownswingBackdrop") = New Chart.Graphics.Backdrop(Color.FromArgb(255,
192, 255))
.Properties("BodyUpswingBackdrop") = New Chart.Graphics.Backdrop(Color.FromArgb(192,
192, 255))
.Properties("WickLine") = New Chart.Graphics.Line(Color.Indigo)
.Properties("BodyWidth") = 7.0F
End With
```

To write the code in C#

C# code

```
Using GrapeCity.ActiveReports.Chart.Graphics
```

C# Code

```
this.ChartControl1.Series[0].Properties["BodyDownswingBackdrop"]= new
Chart.Graphics.Backdrop
(Color.FromArgb(255, 192, 255));
this.ChartControl1.Series[0].Properties["BodyUpswingBackdrop"] = new
Chart.Graphics.Backdrop
(Color.FromArgb(192, 192, 255));
this.ChartControl1.Series(0).Properties("WickLine") = new
Chart.Graphics.Line(Color.Indigo);
this.ChartControl1.Series[0].Properties["BodyWidth"] = 7f;
```

HiLo Chart

A HiLo chart displays stock information using High and Low, or Open and Close, values. The length of the HiLo line is determined by the High and Low values, or the Open and Close values.

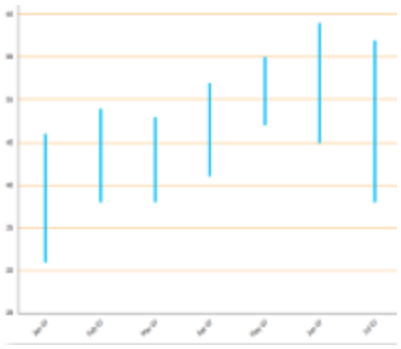


Chart Information	
ChartType	Financial2D
Number of Y values per data point	2
Number of Series	1 or more
Marker Support	Series or Data Point. Marker labels use the first Y value as the default value.
Custom Properties	The **HiloLine** property gets or sets the line information for the HiLo line.

Below is an example of how to set the custom chart properties at run time for a HiLo chart as shown in the image above.

To write the code in Visual Basic.NET

```
Visual Basic
Me.ChartControl1.Series(0).Properties("HiloLine") = New
GrapeCity.ActiveReports.Chart.Graphics.Line(Color.DeepSkyBlue, 4)
```

To write the code in C#

```
C#
this.ChartControl1.Series[0].Properties["HiloLine"] = new
GrapeCity.ActiveReports.Chart.Graphics.Line(Color.DeepSkyBlue, 4);
```

Point and Figure Chart

The point and figure chart uses stacked columns of X's to indicate that demand exceeds supply and columns of O's to indicate that supply exceeds demand to define pricing trends. A new X or O is added to the chart if the price moves higher or lower than the BoxSize value. A new column is added when the price reverses to the level of the BoxSize value multiplied by the ReversalAmount. The use of these values in the point and figure chart to calculate pricing trends makes this chart best suited for long-term financial analysis.

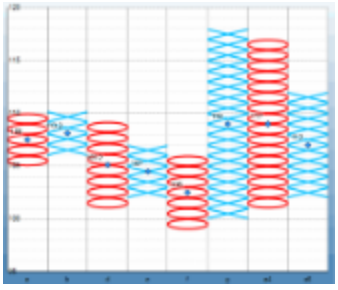


Chart Information	
ChartType	Financial2D
Number of Y values per data point	2
Number of Series	1 or more
Marker Support	Series or Data Points
Custom Properties	The **BoxSize** property gets or sets the amount a price must change in order to create another X or O. The **DownswingLine** property gets or sets the style and color settings for the downswing O's. The **ReversalAmount** property gets or sets the amount that a price must shift in order for a new column to be added. The **UpswingLine** property gets or sets the style and color settings for the upswing X's.

Below is an example of how to set the custom chart properties at run time for a Point and Figure chart.

To write the code in Visual Basic.NET

Visual Basic

```
Imports GrapeCity.ActiveReports.Chart.Graphics
```

Visual Basic

```
With Me.ChartControl1.Series(0)
    .Properties("DownswingLine") = New Chart.Graphics.Line(Color.Red)
    .Properties("UpswingLine") = New Chart.Graphics.Line(Color.Blue)
    .Properties("BoxSize") = 3.0F
End With
```

To write the code in C#

C#

```
using GrapeCity.ActiveReports.Chart.Graphics;
```

C#

```
this.ChartControl1.Series[0].Properties["DownswingLine"] = new
Chart.Graphics.Line(Color.Red);
this.ChartControl1.Series[0].Properties["UpswingLine"] = new
Chart.Graphics.Line(Color.Blue);
this.ChartControl1.Series[0].Properties["BoxSize"] = 3f;
```

Renko Chart

The Renko chart uses bricks of uniform size to chart price movement. When a price moves to a greater or lesser value than

the preset BoxSize value required to draw a new brick, a new brick is drawn in the succeeding column. The change in box color and direction signifies a trend reversal.

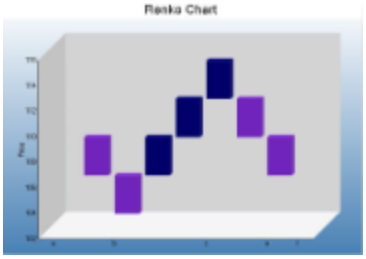


Chart Information	
ChartType	Financial2D
Number of Y values per data point	1
Number of Series	1 or more
Marker Support	Series or Data Points
Custom Properties	The **BodyDownswingBackdrop** property gets or sets the style and color settings for the downswing bricks. The **BodyUpswingBackdrop** property gets or sets the style and color settings for the upswing bricks. The **BoxSize** property gets or sets the amount a price must change in order to create another brick.

Below is an example of how to set the custom chart properties at run time for a Renko chart.

To write the code in Visual Basic.NET

Visual Basic

```
Imports GrapeCity.ActiveReports.Chart.Graphics
```

Visual Basic

```
With Me.ChartControl1.Series(0)
    .Properties("BodyDownswingBackdrop") = New Backdrop(Color.BlueViolet)
    .Properties("BodyUpswingBackdrop") = New Backdrop(Color.Navy)
    .Properties("BoxSize") = 3.0F
End With
```

To write the code in C#

CS

```
using GrapeCity.ActiveReports.Chart.Graphics;
```

CS

```
this.ChartControl1.Series[0].Properties["BodyDownswingBackdrop"] = new
Backdrop(Color.BlueViolet);
```

```
this.ChartControl1.Series[0].Properties["BodyUpswingBackdrop"] = new
Backdrop(Color.Navy);
this.ChartControl1.Series[0].Properties["BoxSize"] = 3f;
```

Kagi Chart

A Kagi chart displays supply and demand trends using a sequence of linked vertical lines. The thickness and direction of the lines vary depending on the price movement. If closing prices go in the direction of the previous Kagi line, then that Kagi line is extended. However, if the closing price reverses by the preset reversal amount, a new Kagi line is charted in the next column in the opposite direction. Thin lines indicate that the price breaks the previous low (supply) while thick lines indicate that the price breaks the previous high (demand).

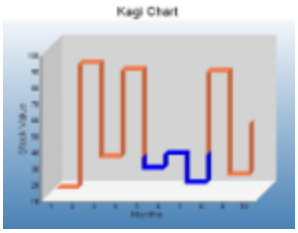


Chart Information	
ChartType	Financial2D
Number of Y values per data point	1
Number of Series	1
Marker Support	Series or Data Points
Custom Properties	The **DownswingLine** property gets or sets the style and color settings to use for a Kagi line which charts a price decrease. The **ReversalAmount** property gets or sets the amount that a price must shift in order for the Kagi line to change direction. The **UpswingLine** property gets or sets the style and color settings to use for a Kagi line which charts a price increase.

Below is an example of how to set the custom chart properties at run time for a Kagi chart.

To write code in Visual Basic.NET

Visual Basic

```
Imports GrapeCity.ActiveReports.Chart.Graphics
```

Visual Basic

```
With Me.ChartControl1.Series(0)
    .Properties("BodyDownswingBackdrop") = New Backdrop(Color.Red)
    .Properties("BodyUpswingBackdrop") = New Backdrop(Color.Blue)
    .Properties("DownswingLine") = New Chart.Graphics.Line(Color.DarkRed)
```

```
    .Properties("ReversalAmount") = "25"
    .Properties("UpswingLine") = New Chart.Graphics.Line(Color.DarkBlue)
    .Properties("Width") = 50.0F
End With
```

To write code in C#

C#

```
using GrapeCity.ActiveReports.Chart.Graphics;
```

C#

```
this.ChartControl1.Series[0].Properties["BodyDownswingBackdrop"] = new
Backdrop(Color.Red);
this.ChartControl1.Series[0].Properties["BodyUpswingBackdrop"] = new
Backdrop(Color.Blue);
this.ChartControl1.Series[0].Properties["DownswingLine"] = new
Chart.Graphics.Line(Color.DarkRed);
this.ChartControl1.Series[0].Properties["ReversalAmount"] = "25";
this.ChartControl1.Series[0].Properties["UpswingLine"] = new
Chart.Graphics.Line(Color.DarkBlue);
this.ChartControl1.Series[0].Properties["Width"] = 50f;
```

Stock Chart

In a stock chart, series are displayed as a set of lines with markers for high, low, close, and open values. Values are represented by the height of the marker as measured by the y-axis. Category labels are displayed on the x-axis.

Stock chart is a visual representation of data related to the stock market. It may be used to represent data like stock prices and stock activities.

Chart Information	
ChartType	Financial2D
Number of Y values per data point	4
Number of Series	1 or more
Marker Support	Series or Data Point
Custom Properties	The **CloseLine** property gets or sets the information for the close value line. The **HiLoLine** property gets or sets the line information for the HiLo line. The **OpenLine** property property gets or sets the information for the open value line. The **TickLen** property property gets or sets the tick length for the close value and open value lines.

Stock Close Only Chart

A Stock chart is a visual representation of data related to the stock market. This type of chart requires four series of values in the correct order (open, high, low, and then close).

Chart Information	
ChartType	Financial2D
Number of Y values per data point	4
Number of Series	1 or more
Marker Support	Series or Data Point
Custom Properties	The **CloseLine** property gets or sets the information for the close value line. The **HiLoLine** property gets or sets the line information for the HiLo line. The **OpenLine** property property gets or sets the information for the open value line. The **TickLen** property property gets or sets the tick length for the close value and open value lines.

Stock Open Only Chart

A Stock chart is a visual representation of data related to the stock market. This type of chart requires four series of values in the correct order (open, high, low, and then close), low, and then close).

Chart Information	
ChartType	Financial2D
Number of Y values per data point	4
Number of Series	3
Marker Support	Series or Data Point
Custom Properties	The **CloseLine** property gets or sets the information for the close value line. The **HiLoLine** property gets or sets the line information for the HiLo line. The **OpenLine** property property gets or sets the information for the open value line. The **TickLen** property property gets or sets the tick length for the close value and open value lines.

Three Line Break Chart

A Three Line Break chart uses vertical boxes or lines to illustrate price changes of an asset or market. Movements are depicted with box colors and styles; movements that continue the trend of the previous box paint similarly while movements that trend oppositely are indicated with a different color and/or style. The opposite trend is only drawn if its value exceeds the extreme value of the previous three boxes or lines. The below Three Line Break depicts upward pricing movement with black boxes and downward pricing movement with red boxes.

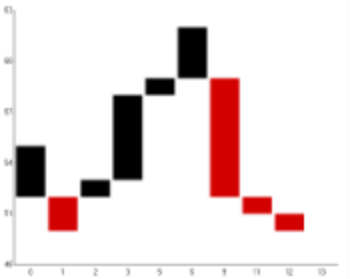


Chart Information	
ChartType	Financial2D
Number of Y values per data point	1
Number of Series	1
Marker Support	Series or Data Points
Chart-Specific Properties	The **BodyDownswingBackdrop** property gets or sets the style and color settings for the downswing boxes. The **BodyUpswingBackdrop** property gets or sets the style and color settings for the upswing boxes. The **NewLineBreak** property gets or sets the number of previous boxes/lines that must be compared before a new box/line is drawn. The default value is 3.

Below is an example of how to set the custom chart properties at run time for a Three Line Break chart.

To write code in Visual Basic.NET

Visual Basic

```
Imports GrapeCity.ActiveReports.Chart.Graphics
```

Visual Basic

```
With Me.ChartControl1.Series(0)
    .Properties("BodyDownswingBackdrop") = New Backdrop(Color.Red)
    .Properties("BodyUpswingBackdrop") = New Backdrop(Color.Black)
    .Properties("NewLineBreak") = 3
End With
```

To write code in C#

C#

```
using GrapeCity.ActiveReports.Chart.Graphics;
```

C#

```
this.ChartControl1.Series[0].Properties["BodyDownswingBackdrop"] = new
Backdrop(Color.Red);
this.ChartControl1.Series[0].Properties["BodyUpswingBackdrop"] = new
Backdrop(Color.Black);
this.ChartControl1.Series[0].Properties["NewLineBreak"] = 3;
```

3D Financial Charts

Given below is the list of 3D charts that fall under the Financial Chart category.

Kagi Chart

A Kagi chart displays supply and demand trends using a sequence of linked vertical lines. The thickness and direction of the lines vary depending on the price movement. If closing prices go in the direction of the previous Kagi line, then that Kagi line is extended. However, if the closing price reverses by the preset reversal amount, a new Kagi line is charted in the next column in the opposite direction. Thin lines indicate that the price breaks the previous low (supply) while thick lines indicate that the price breaks the previous high (demand).

Chart Information	
ChartType	Financial3D
Number of Y values per data point	1
Number of Series	1
Marker Support	Series or Data Points
Custom Properties	The **BodyDownswingBackdrop** property gets or sets the style and color settings for the three-dimensional side view of downswing Kagi lines. This property is only available with the Kagi 3D chart type, and is only effective when the Width property is set to a value higher than 25. The **BodyUpswingBackdrop** property gets or sets the style and color settings for the three-dimensional side view of upswing Kagi lines. This property is only available with the Kagi 3D chart type, and is only effective when the Width property is set to a value higher than 25. The **DownswingLine** property gets or sets the style and color settings to use for a Kagi line which charts a price decrease. The **ReversalAmount** property gets or sets the amount that a price must shift in order for the Kagi line to change direction. The **UpswingLine** property gets or sets the style and color settings to use for a Kagi line which charts a price increase. The **Width** property gets or sets the width of the three-dimensional side view of the Kagi lines.This property is only available with the Kagi 3D chart type, and must be set higher than its default value of 1 in order to display body downswing and upswing backdrops.

Below is an example of how to set the custom chart properties at run time for a Kagi chart.

To write code in Visual Basic.NET

Visual Basic

```
Imports GrapeCity.ActiveReports.Chart.Graphics
```

Visual Basic

```
With Me.ChartControl1.Series(0)
```

```
    .Properties("BodyDownswingBackdrop") = New Backdrop(Color.Red)
    .Properties("BodyUpswingBackdrop") = New Backdrop(Color.Blue)
    .Properties("DownswingLine") = New Chart.Graphics.Line(Color.DarkRed)
    .Properties("ReversalAmount") = "25"
    .Properties("UpswingLine") = New Chart.Graphics.Line(Color.DarkBlue)
    .Properties("Width") = 50.0F
End With
```

To write code in C#

C#

```
using GrapeCity.ActiveReports.Chart.Graphics;
```

C#

```
this.ChartControl1.Series[0].Properties["BodyDownswingBackdrop"] = new
Backdrop(Color.Red);
this.ChartControl1.Series[0].Properties["BodyUpswingBackdrop"] = new
Backdrop(Color.Blue);
this.ChartControl1.Series[0].Properties["DownswingLine"] = new
Chart.Graphics.Line(Color.DarkRed);
this.ChartControl1.Series[0].Properties["ReversalAmount"] = "25";
this.ChartControl1.Series[0].Properties["UpswingLine"] = new
Chart.Graphics.Line(Color.DarkBlue);
this.ChartControl1.Series[0].Properties["Width"] = 50f;
```

Renko Chart

The Renko chart uses bricks of uniform size to chart price movement. When a price moves to a greater or lesser value than the preset BoxSize value required to draw a new brick, a new brick is drawn in the succeeding column. The change in box color and direction signifies a trend reversal.

Chart Information	
ChartType	Financial3D
Number of Y values per data point	1
Number of Series	1 or more
Marker Support	Series or Data Points
	The **BodyUpswingBackdrop** property gets or sets the style and color settings for the upswing bricks.

Custom Properties	**BodyDownswingBackdrop** Gets or sets the style and color settings for the downswing bricks. The **BoxSize** property gets or sets the amount a price must change in order to create another brick.

Below is an example of how to set the custom chart properties at run time for a Renko chart.

To write code in Visual Basic.NET

Visual Basic

```
Imports GrapeCity.ActiveReports.Chart.Graphics
```

Visual Basic

```
With Me.ChartControl1.Series(0)
    .Properties("BodyDownswingBackdrop") = New Backdrop(Color.Red)
    .Properties("BodyUpswingBackdrop") = New Backdrop(Color.Blue)
    .Properties("BoxSize") = 3.0F
End With
```

To write code in C#

CS

```
using GrapeCity.ActiveReports.Chart.Graphics;
```

C#

```
this.ChartControl1.Series[0].Properties["BodyDownswingBackdrop"] = new
Backdrop(Color.Red);
this.ChartControl1.Series[0].Properties["BodyUpswingBackdrop"] = new
Backdrop(Color.Blue);
this.ChartControl1.Series[0].Properties["BoxSize"] = 3f;
```

Three Line Break Chart

Three line break 3D chart is a chart rendered in 3D.

Chart Information	
ChartType	Financial3D
Number of Y values per data point	1
Number of Series	1
Marker Support	Series or Data Points

Custom Properties	The **BodyDownswingBackdrop** property gets or sets the style and color settings for the downswing boxes. The **BodyUpswingBackdrop** property gets or sets the style and color settings for the upswing boxes. The **NewLineBreak** property gets or sets the number of previous boxes/lines that must be compared before a new box/line is drawn. The default value is 3.

Point and Bubble Charts

Point or Bubble charts represent data by means of points and bubbles.

The ActiveReports Chart control can draw a number of point/bubble chart types:

- Bubble, BubbleXY, PlotXY and Scatter.

2D Point/Bubble Charts

This section describes 2D charts that fall under the Point/Bubble Chart category.

2D Point/Bubble Charts

Given below is the list of 2D charts that fall under the Point/Bubble Chart category.

Bubble Chart

The Bubble chart is an XY chart in which bubbles represent data points. The first Y value is used to plot the bubble along the Y axis, and the second Y value is used to set the size of the bubble. The bubble shape can be changed using the series Shape property.

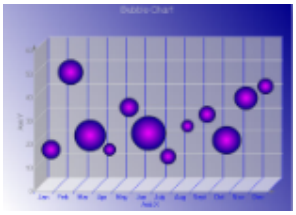


Chart Information	
ChartType	Point/Bubble2D
Number of Y values per data point	2
Number of Series	1 or more
Marker Support	Series or Data Point. Marker labels use the second Y value as the default value.
Custom Properties	The **MaxSizeFactor** property gets or sets the maximum size of the bubble radius. Values must be less than or equal to 1. Default is .25. The **MaxValue** property gets or sets the bubble size that is used as the maximum. The **MinValue** property gets or sets the bubble size that is used as the minimum. The **Shape** property gets or sets the shape of the bubbles. Uses or returns a valid MarkerStyle enumeration value.

Below is an example of setting the custom chart properties at run time for a bubble chart as shown in the image above.

To write code in Visual Basic.NET

```
Visual Basic
Me.ChartControl1.Series(0).Properties("MaxSizeFactor") = 0.25F
Me.ChartControl1.Series(0).Properties("MaxValue") = 55.0R
Me.ChartControl1.Series(0).Properties("MinValue") = 5.0R
Me.ChartControl1.Series(0).Properties("Shape") =
GrapeCity.ActiveReports.Chart.MarkerStyle.Circle
```

To write code in C#

```
C#
this.ChartControl1.Series[0].Properties["MaxSizeFactor"] = .25f;
this.ChartControl1.Series[0].Properties["MaxValue"] = 55D;
this.ChartControl1.Series[0].Properties["MinValue"] = 5D;
this.ChartControl1.Series[0].Properties["Shape"] =
GrapeCity.ActiveReports.Chart.MarkerStyle.Circle;
```

Bubble XY Chart

The Bubble XY chart is an XY chart in which bubbles represent data points. The BubbleXY uses a numerical X axis and plots the x values and first set of Y values on the chart. The second Y value is used to set the size of the bubble.

Chart Information	
ChartType	Point/Bubble2D
Number of Y values per data point	2
Number of Series	1 or more
Marker Support	Series or Data Point. Marker labels use the second Y value as the default value.
Custom Properties	The **MaxSizeFactor** gets or sets the maximum size of the bubble radius. Values must be less than or equal to 1. Default is .25. The **MaxValue** property gets or sets the bubble size that is used as the maximum. The **MinValue** property gets or sets the bubble size that is used as the minimum. The **Shape** property gets or sets the shape of the bubbles. Uses or returns a valid MarkerStyle enumeration value.

Below is an example of setting the custom chart properties at run time for a bubble XY chart as shown in the image above.

To write code in Visual Basic.NET

```
Visual Basic
Me.ChartControl1.Series(0).Properties("MaxSizeFactor") = 0.25F
Me.ChartControl1.Series(0).Properties("MaxValue") = 50.0R
Me.ChartControl1.Series(0).Properties("MinValue") = 0.0R
```

```
Me.ChartControl1.Series(0).Properties("Shape") =
GrapeCity.ActiveReports.Chart.MarkerStyle.InvTriangle
```

To write code in C#

C#

```
this.ChartControl1.Series[0].Properties["MaxSizeFactor"] = .25f;
this.ChartControl1.Series[0].Properties["MinValue"] = 0D;
this.ChartControl1.Series[0].Properties["MaxValue"] = 50D;
this.ChartControl1.Series[0].Properties["Shape"] =
GrapeCity.ActiveReports.Chart.MarkerStyle.InvTriangle;
```

Plot XY Chart

A plot XY chart shows the relationships between numeric values in two or more series sets of XY values.

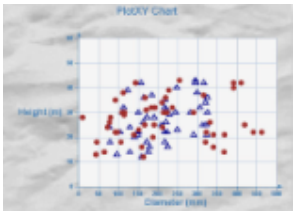


Chart Information	
ChartType	Point/Bubble2D
Number of Y values per data point	1
Number of Series	1 or more
Marker Support	Series or Data Point
Custom Properties	None

Scatter Chart

Use a scatter chart to compare values across categories.

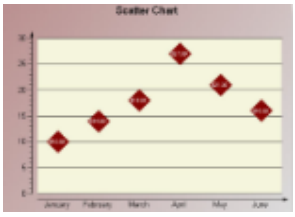
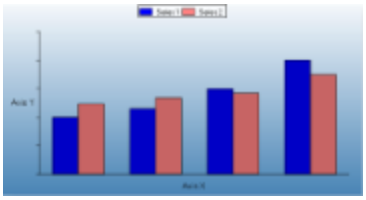


Chart Information

ChartType	Point/Bubble2D
Number of Y values per data point	1
Number of Series	1 or more
Marker Support	Series or Data Point
Custom Properties	None

Chart Series

A chart series is the key to showing data in a chart. All data is plotted in a chart as a series and all charts contain at least one series. The bars in the image below depict two series in a simple bar chart.

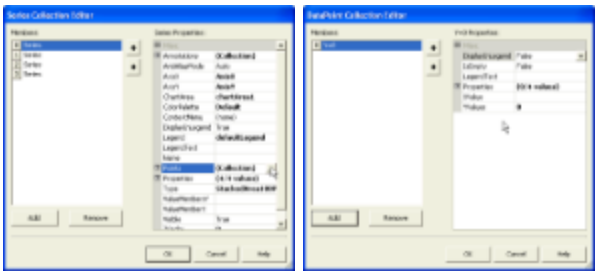


Each series is made up of a set of data points consisting of an X value that determines where on the X axis the data is plotted, and one or more Y values. Most charts use one Y value but a few charts such as the Bubble, BubbleXY, and the financial charts take multiple Y values.

When you bind data to a series, the X value is bound using the ValueMembersX property on the Series object, and the Y value is bound using the ValueMembersY property.

The Series object also contains properties for each individual series, including chart type, custom chart properties, annotations, containing chart area, and more. Each chart type in the ActiveReports Chart control contains series-specific properties that apply to it. You can set the chart type and these series-specific properties in the Series Collection Editor dialog, which opens when you click the ellipsis button next to the **Series (Collection)** property in the Visual Studio Properties window.

You can manipulate each data point in the DataPoint Collection dialog box. You can access the dialog from the Series Collection Editor by clicking the ellipsis button next to the **Points (Collection)** property.



When you set a property on the Series object, it is applied to all data point objects in the series unless a different value for the property is set on a specific data point. In that case, the data point property setting overrides the series property setting for that particular data point. Note that for charts bound to a data source, you do not have access to the DataPoint collection in the dialog.

If you specify the value for any of the custom properties, this value is not cleared when you change the ChartType. Although this will show properties that do not apply to certain ChartTypes, it has the advantage of keeping your settings in case you accidentally change the ChartType.

Setting chart and series-specific properties at run time

To set custom properties for a chart on the series programmatically, reference the series by name or index and use the string Properties attribute name you wish to set.

The following code samples set the shape for bubbles on a bubble chart to diamond.

To write code in Visual Basic.Net

Visual Basic.NET code. Paste INSIDE the section Format event.

```
Me.ChartControl1.Series(0).Properties("Shape") = Chart.MarkerStyle.Diamond
```

To write the code in C#

C# code. Paste INSIDE the section Format event.

```
this.chartControl1.Series[0].Properties["Shape"] =
GrapeCity.ActiveReports.Chart.MarkerStyle.Diamond;
```

To set custom properties for a chart on the data points object programmatically, reference the series by name or index, reference the data point by index, and use the string Properties attribute name you wish to set.

The following code samples set the explode factor on a doughnut chart for the second point in the series.

To write code in Visual Basic.Net

Visual Basic.NET code. Paste INSIDE the section Format event.

```
Me.ChartControl1.Series(0).Points(1).Properties("ExplodeFactor") = 0.5F
```

To write the code in C#

C# code. Paste INSIDE the section Format event.

```
this.chartControl1.Series[0].Points[1].Properties["ExplodeFactor"] = .5f;
```

Chart Appearance

The following section explains in what ways you can modify the chart appearance.

Chart Effects
: This section describes the visual effects that are available for the Chart data region.

Chart Control Items
: This section describes the chart elements that you can use to customize the Chart data region.

Chart Axes and Walls
: This section provides basic information about the axes and walls in the Chart data region.

Chart Effects

These topics introduce some basic information on the visual effects of the Chart data region.

Colors
: Learn about the different ways you can change the color and gradients to enhance the visual appearance of a chart.

3D Effect
: This section describes 3D effects that you can use to customize the chart.

Alpha Blending
: This section explains about alpha blending property of the chart.

Lightning
: Learn about directional light ratio, line type and line source of the chart.

Colors

In the Chart data region, colors can be used in different ways to enhance the chart's appearance, distinguish different series, point out or draw attention to data information such as averages, and more.

Color Palettes

The Chart data region includes several pre-defined color palettes that can be used to automatically set the colors for data values in a series. The pre-defined palettes are as follows.

- **Default** The default palette.
- **Cascade** A cascade of eight cool colors ranging from deep teal down through pale orchid.
- **Springtime** The colors of spring, in deep green, two vivid colors and five pastels.
- **Iceberg** A range of the soft blues and greys found in an iceberg.
- **Confetti** A sprinkling of bright and pastel colors.
- **Greens** A palette of greens.
- **Berries** The colors of berries.
- **Autumn** A quiet palette of autumn colors.
- **Murphy** A range of the soft blues and greens.

These enumerated values are accessed through the Series class with code like the following.

To write code in Visual Basic.NET

Visual Basic

```
Me.ChartControl1.Series(0).ColorPalette = Chart.ColorPalette.Iceberg
```

To write code in C#

C#

```
this.ChartControl1.Series[0].ColorPalette =
GrapeCity.ActiveReports.Chart.ColorPalette.Iceberg;
```

Gradients

Gradients can be used in object backdrops to enhance the visual appearance of various chart items. Gradients can be used in the following chart sections:

- Chart
- Chart area
- Wall
- Title
- Legend
- Legend item (for custom legend items)
- WallRange
- Series
- Data point
- Marker
- Marker Label
- Annotation TextBar

You can set gradients for a backdrop at run time by creating a **BackdropItem**, setting its **Style** property to Gradient, setting the **GradientType**, and setting the two colors to use for the gradient as shown in the following example.

To write code in Visual Basic.NET

Visual Basic

```
Imports GrapeCity.ActiveReports.Chart.Graphics
```

Visual Basic

```
Dim bItem As New GrapeCity.ActiveReports.Chart.BackdropItem
bItem.Style = Chart.Graphics.BackdropStyle.Gradient
bItem.Gradient = Chart.Graphics.GradientType.Vertical
bItem.Color = Color.Purple
bItem.Color2 = Color.White
Me.ChartControl1.Backdrop = bItem
```

To write code in C#

C#

```
using GrapeCity.ActiveReports.Chart.Graphics;
```

C#

```
GrapeCity.ActiveReports.Chart.BackdropItem bItem = new
GrapeCity.ActiveReports.Chart.BackdropItem();
bItem.Style = GrapeCity.ActiveReports.Chart.Graphics.BackdropStyle.Gradient;
bItem.Gradient = GrapeCity.ActiveReports.Chart.Graphics.GradientType.Vertical;
bItem.Color = System.Drawing.Color.Purple;
bItem.Color2 = System.Drawing.Color.White;
this.ChartControl1.Backdrop = bItem;
```

3D Effects

Using the projection and viewpoint settings, you can display your 3D chart at any angle to provide the desired view or call attention to a specific chart section.

Projection

Determine the projection for a 3D chart using the following properties.

Property Name	Description
ZDepthRatio	The Z depth ratio is the level of depth the Z axis has in the chart. The ratio of the length specified in the X-axis of Z-axis Values range from 0 (for a 2D chart) to 1.0. This property is useful in adjusting the size of 3D chart.
ProjectionDX	The origin position of the Z axis in relation to the X axis. This property is valid only when the ProjectionType is Orthogonal.
ProjectionDY	The origin position of the Z axis in relation to the Y axis. This property is valid only when the ProjectionType is Orthogonal.
ProjectionType	The type of projection used for the chart. In order to show charts three dimensionally, the ProjectionType in the ChartArea Collection editor must be set to Orthogonal. To access this dialog box, click the ellipsis button next to the ChartAreas (Collection) property in the Properties Window.
HorizontalRotation	The HorizontalRotation property allows you to set the degree (-90° to 90°) of horizontal rotation from which the chart is seen.
VerticalRotation	The VerticalRotation property allows you to set the degree (-90° to 90°) of vertical rotation from which the chart is seen.

Alpha Blending

The Backdrop class in the Chart control has an Alpha property which employs GDI+, and is used to set the transparency level of each object's backdrop. GDI+ uses 32 bits overall and 8 bits per alpha, red, green, and blue channels respectively to indicate the transparency and color of an object. Like a color channel's levels of color, the alpha channel represents 256 levels of transparency.

The default value of the Alpha property is 255, which represents a fully opaque color. For a fully transparent color, set this value to 0. To blend the color of the object's backdrop with the background color, use a setting between 0 and 255.

In the Chart control, you can use the Color.FromArgb method provided by standard Color constructor of .NET Framework class library to set the alpha and color levels for a particular chart element.

The following example shows how you can use the method to set the alpha and color values for the chart backdrop.

To write code in Visual Basic.NET

Visual Basic

```
Me.ChartControl1.Backdrop = New
GrapeCity.ActiveReports.Chart.BackdropItem(Color.FromArgb(100, 0, 11, 220))
```

To write code in C#

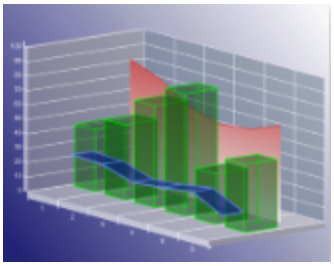
C#

```
this.ChartControl1.Backdrop = new
GrapeCity.ActiveReports.Chart.BackdropItem(System.Drawing.Color.FromArgb(100, 0, 11,
220));
```

Changing the alpha level of a chart element reveals other items that are beneath the object. Because you can set the alpha level for any chart element that supports color, you can also create custom effects for any chart. For example, you can use alpha blending to combine background images with a semi-transparent chart backdrop to create a watermark look.

Lighting

The Chart control allows you to completely customize lighting options for 3D charts.



Directional Light Ratio

Using the DirectionalLightRatio property, you can control the directional or ambient intensity ratio.

Light Type

By setting the **Type** property to one of the enumerated LightType values, you can control the type of lighting used in the chart. The settings are as follows:

- **Ambient**: An ambient light source is used. It is equal to DirectionalLightRatio = 0.
- **Directional**: An infinite directional light source (like the sun, the light source having the same angle of incidence from each side) is used.
- **Point**: A point light source (light is generated from one point and light source having different angle of incidence according to the sides) is used.

Light Source

You can also set the **Source** property to a Point3d object, which controls the location of the light source.

Chart Control Items

The following topics review the chart control items that you can use to customize your chart.

Chart Annotation
: This topic explains how you can add annotations to your chart.

Chart Titles and Footers
: This topic explains how you can add a title or a footer to your chart.

Legends
: This topic explains how you can add legends to your chart.

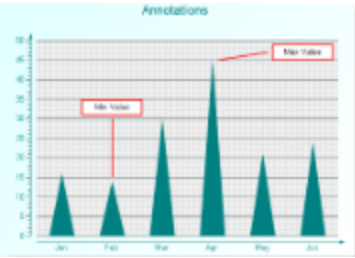
Markers
: This topic explains how to create markers for showing specific data series values.

Label Symbols
: This topic describes how to set format strings to display data in a legend, a marker or a symbol to be used as placeholders.

Constant Lines and Stripes
: This topic explains how to add constant lines and stripes to your chart.

Chart Annotations

The Chart control offers a built-in annotation tool to allow you to include floating text bars or images in your charts or call attention to specific items or values in your charts using the line and text bar controls included in the Annotations Collection Editor. You can find the Annotations Collection Editor under the Series properties of the Chart Designer.



The following properties are important while setting up annotations for your chart:

- **StartPoint** Sets the starting point (X and Y axis values) for an annotation line.
- **EndPoint** Sets the end point (X and Y axis values) for an annotation line.
- **AnchorPlacement** Sets the position of the anchor point for the text bar on the chart surface.
- **AnchorPoint** Sets the point (X and Y axis values) where the text bar will be anchored based on the anchor placement selected.

The following code demonstrates creating annotation lines and annotation text, setting their properties, and adding them to the series annotations collection at run time.

To write code in Visual Basic.NET

Visual Basic

```
' create the annotation lines and text bar.
Dim aLine1 As New GrapeCity.ActiveReports.Chart.Annotations.AnnotationLine
Dim aLine2 As New GrapeCity.ActiveReports.Chart.Annotations.AnnotationLine
Dim aText1 As New GrapeCity.ActiveReports.Chart.Annotations.AnnotationTextBar
Dim aText2 As New GrapeCity.ActiveReports.Chart.Annotations.AnnotationTextBar

' set the properties for each line and text bar.
With aLine1
        .EndPoint = New GrapeCity.ActiveReports.Chart.Graphics.Point2d(1.5F, 30.0F)
        .Line = New
GrapeCity.ActiveReports.Chart.Graphics.Line(System.Drawing.Color.Red, 2)
        .StartPoint = New GrapeCity.ActiveReports.Chart.Graphics.Point2d(1.5F, 15.0F)
End With
With aLine2
        .EndPoint = New GrapeCity.ActiveReports.Chart.Graphics.Point2d(4.6F, 47.0F)
        .Line = New
GrapeCity.ActiveReports.Chart.Graphics.Line(System.Drawing.Color.Red, 2)
        .StartPoint = New GrapeCity.ActiveReports.Chart.Graphics.Point2d(3.6F, 45.0F)
End With
With aText1
        .AnchorPlacement =
GrapeCity.ActiveReports.Chart.Annotations.AnchorPlacementType.Bottom
        .AnchorPoint = New GrapeCity.ActiveReports.Chart.Graphics.Point2d(1.5F, 31.0F)
        .Height = 25.0F
        .Line = New
GrapeCity.ActiveReports.Chart.Graphics.Line(System.Drawing.Color.Red, 2)
        .Text = "Min Value"
        .Width = 100.0F
End With
With aText2
        .AnchorPlacement =
GrapeCity.ActiveReports.Chart.Annotations.AnchorPlacementType.Left
        .AnchorPoint = New GrapeCity.ActiveReports.Chart.Graphics.Point2d(4.7F, 47.0F)
        .Height = 25.0F
        .Line = New
GrapeCity.ActiveReports.Chart.Graphics.Line(System.Drawing.Color.Red, 2)
        .Text = "Max Value"
        .Width = 100.0F
End With

' add the annotation lines and text bars to the annotations collection for the series
Me.ChartControl1.Series(0).Annotations.AddRange(New
GrapeCity.ActiveReports.Chart.Annotations.Annotation() {aLine1, aLine2, aText1, aText2})
```

To write code in C#

C#

```
// create the annotation lines and text bar.
GrapeCity.ActiveReports.Chart.Annotations.AnnotationLine aLine1 = new
GrapeCity.ActiveReports.Chart.Annotations.AnnotationLine();
GrapeCity.ActiveReports.Chart.Annotations.AnnotationLine aLine2 = new
GrapeCity.ActiveReports.Chart.Annotations.AnnotationLine();
GrapeCity.ActiveReports.Chart.Annotations.AnnotationTextBar aText1 = new
GrapeCity.ActiveReports.Chart.Annotations.AnnotationTextBar();
GrapeCity.ActiveReports.Chart.Annotations.AnnotationTextBar aText2 = new
GrapeCity.ActiveReports.Chart.Annotations.AnnotationTextBar();

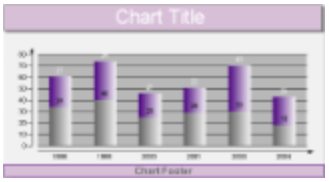
// set the properties for each line and text bar.
aLine1.EndPoint = new GrapeCity.ActiveReports.Chart.Graphics.Point2d(1.5F, 30F);
aLine1.Line = new GrapeCity.ActiveReports.Chart.Graphics.Line(System.Drawing.Color.Red,
2);
aLine1.StartPoint = new GrapeCity.ActiveReports.Chart.Graphics.Point2d(1.5F, 15F);
aLine2.EndPoint = new GrapeCity.ActiveReports.Chart.Graphics.Point2d(4.6F, 47F);
aLine2.Line = new GrapeCity.ActiveReports.Chart.Graphics.Line(System.Drawing.Color.Red,
2);
aLine2.StartPoint = new GrapeCity.ActiveReports.Chart.Graphics.Point2d(3.6F, 45F);
aText1.AnchorPlacement =
GrapeCity.ActiveReports.Chart.Annotations.AnchorPlacementType.Bottom;
aText1.AnchorPoint = new GrapeCity.ActiveReports.Chart.Graphics.Point2d(1.5F, 31F);
aText1.Height = 25F;
aText1.Line = new GrapeCity.ActiveReports.Chart.Graphics.Line(System.Drawing.Color.Red,
2);
aText1.Text = "Min Value";
aText1.Width = 100F;
aText2.AnchorPlacement =
GrapeCity.ActiveReports.Chart.Annotations.AnchorPlacementType.Left;
aText2.AnchorPoint = new GrapeCity.ActiveReports.Chart.Graphics.Point2d(4.7F, 47F);
aText2.Height = 25F;
aText2.Line = new GrapeCity.ActiveReports.Chart.Graphics.Line(System.Drawing.Color.Red,
2);
aText2.Text = "Max Value";
aText2.Width = 100F;

// add the annotation lines and text bars to the annotations collection for the series
this.ChartControl1.Series[0].Annotations.AddRange(
new GrapeCity.ActiveReports.Chart.Annotations.Annotation[] {aLine1, aLine2, aText1,
aText2});
```

Chart Titles and Footers

The Chart control allows you to add custom titles to your charts. The Titles collection is accessible from the Chart object. With the ability to add as many titles as needed, dock them to any side of a chart area, change all of the font properties,

add borders and shadows, make the background look the way you want it, and change the location of the text, you can easily make your titles look the way you want them to look.



The following code demonstrates creating header and footer titles, setting their properties, and adding them to the titles collection at run time.

1. In design view of the report, double-click the section where you placed your chart. This creates a Format event handling method for the section.
2. Add code to the handler to create header and footer titles.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the section Format event.

```
' create the header and footer titles
Dim tHeader As New GrapeCity.ActiveReports.Chart.Title
Dim tFooter As New GrapeCity.ActiveReports.Chart.Title

' set the properties for the header
tHeader.Alignment = Chart.Alignment.Center
tHeader.Backdrop = New
GrapeCity.ActiveReports.Chart.Graphics.Backdrop(System.Drawing.Color.Thistle)
tHeader.Border = New GrapeCity.ActiveReports.Chart.Border(New
GrapeCity.ActiveReports.Chart.Graphics.Line(System.Drawing.Color.DimGray), 3)
tHeader.DockArea = Me.ChartControl1.ChartAreas(0)
tHeader.Docking = Chart.DockType.Top
tHeader.Font = New GrapeCity.ActiveReports.Chart.FontInfo(System.Drawing.Color.White,
New System.Drawing.Font("Arial", 25.0F))
tHeader.Text = "Chart Title"
tHeader.Visible = True

' set the properties for the footer
tFooter.Alignment = Chart.Alignment.Center
tFooter.Backdrop = New
GrapeCity.ActiveReports.Chart.Graphics.Backdrop(System.Drawing.Color.Thistle)
tFooter.Border = New GrapeCity.ActiveReports.Chart.Border(New
GrapeCity.ActiveReports.Chart.Graphics.Line(System.Drawing.Color.Indigo), 0,
System.Drawing.Color.Black)
tFooter.DockArea = Me.ChartControl1.ChartAreas(0)
tFooter.Docking = Chart.DockType.Bottom
tFooter.Font = New GrapeCity.ActiveReports.Chart.FontInfo(System.Drawing.Color.DimGray,
New System.Drawing.Font("Arial", 12.0F, System.Drawing.FontStyle.Bold))
tFooter.Text = "Chart Footer"
tFooter.Visible = True
```

```
' add the header and footer titles to the titles collection
Me.ChartControl1.Titles.AddRange(New GrapeCity.ActiveReports.Chart.Title() {tHeader,
tFooter})
```

To write the code in C#

C# code. Paste INSIDE the section Format event.

```
// create the header and footer titles
GrapeCity.ActiveReports.Chart.Title tHeader = new GrapeCity.ActiveReports.Chart.Title();
GrapeCity.ActiveReports.Chart.Title tFooter = new GrapeCity.ActiveReports.Chart.Title();

// set the properties for the header
tHeader.Alignment = Chart.Alignment.Center;
tHeader.Backdrop = new
GrapeCity.ActiveReports.Chart.Graphics.Backdrop(System.Drawing.Color.Thistle);
tHeader.Border = new GrapeCity.ActiveReports.Chart.Border(new
GrapeCity.ActiveReports.Chart.Graphics.Line(System.Drawing.Color.DimGray), 3);
tHeader.DockArea = this.ChartControl1.ChartAreas[0];
tHeader.Docking = Chart.DockType.Top;
tHeader.Font = new GrapeCity.ActiveReports.Chart.FontInfo(System.Drawing.Color.White,
new System.Drawing.Font("Arial", 25F));
tHeader.Text = "Chart Title";
tHeader.Visible = true;

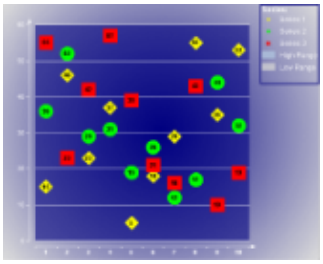
// set the properties for the footer
tFooter.Alignment = Chart.Alignment.Center;
tFooter.Backdrop = new
GrapeCity.ActiveReports.Chart.Graphics.Backdrop(System.Drawing.Color.Thistle);
tFooter.Border = new GrapeCity.ActiveReports.Chart.Border(new
GrapeCity.ActiveReports.Chart.Graphics.Line(System.Drawing.Color.Indigo), 0,
System.Drawing.Color.Black);
tFooter.DockArea = this.ChartControl1.ChartAreas[0];
tFooter.Docking = Chart.DockType.Bottom;
tFooter.Font = new GrapeCity.ActiveReports.Chart.FontInfo(System.Drawing.Color.DimGray,
new System.Drawing.Font("Arial", 12F, System.Drawing.FontStyle.Bold));
tFooter.Text = "Chart Footer";
tFooter.Visible = true;

// add the header and footer titles to the titles collection
this.ChartControl1.Titles.AddRange(new GrapeCity.ActiveReports.Chart.Title[]
{tHeader,tFooter});
```

Legends

The Chart control automatically creates a legend item for each series added to a chart at design time and sets the **Legend**

property for each series by default. However, the legend's **Visible** property must be set to True for getting it displayed with the chart. The text for legend caption is taken from the **Name** property on the series.



Note: Each Series displayed in the Legend must have a Name. If the Name property is not set, the Series does not show up in the Legend.

The following code demonstrates how to create a legend at run time, add it to the Legends collection of the Chart object and set the legend property of the series to the new legend, resulting in the legend shown above.

To write code in Visual Basic.NET

Visual Basic code. Paste INSIDE the section Format event

```
' create the legend and title for the legend
Dim legend1 As New GrapeCity.ActiveReports.Chart.Legend
Dim lHeader As New GrapeCity.ActiveReports.Chart.Title

' set the properties for the legend title
lHeader.Backdrop = New
GrapeCity.ActiveReports.Chart.Graphics.Backdrop(Chart.Graphics.BackdropStyle.Transparent,
 _Color.White, Color.White, Chart.Graphics.GradientType.Vertical,
 Drawing2D.HatchStyle.DottedGrid, Nothing, _Chart.Graphics.PicturePutStyle.Stretched)
lHeader.Border = New GrapeCity.ActiveReports.Chart.Border(New
Chart.Graphics.Line(Color.White, 2,
 _Chart.Graphics.LineStyle.None), 0, Color.Black)
lHeader.Font = New GrapeCity.ActiveReports.Chart.FontInfo
(Color.White, New System.Drawing.Font("Arial", 10.0F, _FontStyle.Bold))
lHeader.Text = "Series:"

' set the properties for the legend and add it to the legends collection
legend1.Alignment = GrapeCity.ActiveReports.Chart.Alignment.TopRight
legend1.Backdrop = New
GrapeCity.ActiveReports.Chart.BackdropItem(Chart.Graphics.BackdropStyle.Transparent,
 _Color.Gray, Color.White, Chart.Graphics.GradientType.Vertical,
Drawing2D.HatchStyle.DottedGrid, Nothing,
 _Chart.Graphics.PicturePutStyle.Stretched)
legend1.Border = New GrapeCity.ActiveReports.Chart.Border(New Chart.Graphics.Line(Color.Navy,
2), _0, Color.Black)
legend1.DockArea = Me.ChartControl1.ChartAreas(0)
legend1.LabelsFont = New GrapeCity.ActiveReports.Chart.FontInfo(Color.White, New
System.Drawing.Font("Arial", 9.0F))
legend1.Header = lHeader
legend1.MarginX = 5
legend1.MarginY = 5
```

```
Me.ChartControl1.Legends.Add(legend1)

' Generate legend items
Dim legenditem As New GrapeCity.ActiveReports.Chart.LegendItem
Dim legenditem2 As New GrapeCity.ActiveReports.Chart.LegendItem

' Set properties of legend item
legenditem.Text = "High Range"
legenditem2.Text = "Low Range"
legenditem.Marker.Style = Chart.MarkerStyle.None
legenditem2.Marker.Style = Chart.MarkerStyle.None
legenditem.Border.Line.Style = Chart.Graphics.LineStyle.None
legenditem2.Border.Line.Style = Chart.Graphics.LineStyle.None
legenditem.Backdrop = New GrapeCity.ActiveReports.Chart.BackdropItem
(System.Drawing.Color.LightSteelBlue, Chart.Graphics.AntiAliasMode.Auto)
legenditem2.Backdrop = New GrapeCity.ActiveReports.Chart.BackdropItem
(System.Drawing.Color.LightGray, Chart.Graphics.AntiAliasMode.Auto)

' Add to legend collection
legend1.LegendItems.Add(legenditem)
legend1.LegendItems.Add(legenditem2)

' Set the legend property of the series to the legend you created
Me.ChartControl1.Series(0).Legend = legend1
Me.ChartControl1.Series(1).Legend = legend1
Me.ChartControl1.Series(2).Legend = legend1
```

To write code in C#

C# code. Paste INSIDE the section Format event

```
// create the legend and title for the legend
GrapeCity.ActiveReports.Chart.Legend legend1 = new GrapeCity.ActiveReports.Chart.Legend();
GrapeCity.ActiveReports.Chart.Title lHeader = new GrapeCity.ActiveReports.Chart.Title();

// set the properties for the legend title
lHeader.Backdrop = new GrapeCity.ActiveReports.Chart.Graphics.Backdrop
(GrapeCity.ActiveReports.Chart.Graphics.BackdropStyle.Transparent,System.Drawing.Color.White,
System.Drawing.Color.White, GrapeCity.ActiveReports.Chart.Graphics.GradientType.Vertical,
System.Drawing.Drawing2D.HatchStyle.DottedGrid, null, GrapeCity.ActiveReports.Chart.Graphics.
PicturePutStyle.Stretched);

lHeader.Border = new GrapeCity.ActiveReports.Chart.Border(new
GrapeCity.ActiveReports.Chart.Graphics.Line
(System.Drawing.Color.White, 2, GrapeCity.ActiveReports.Chart.Graphics.LineStyle.None), 0,
 System.Drawing.Color.Black);

lHeader.Font = new GrapeCity.ActiveReports.Chart.FontInfo(System.Drawing.Color.White,
new System.Drawing.Font("Arial", 10.0F, System.Drawing.FontStyle.Bold));
lHeader.Text = "Series:";
```

```
// set the properties for the legend and add it to the legends collection.
legend1.Alignment = GrapeCity.ActiveReports.Chart.Alignment.TopRight;
legend1.Backdrop = new GrapeCity.ActiveReports.Chart.BackdropItem
(GrapeCity.ActiveReports.Chart.Graphics.BackdropStyle.Transparent, System.Drawing.Color.Gray,
System.Drawing.Color.White, GrapeCity.ActiveReports.Chart.Graphics.GradientType.Vertical,
System.Drawing.Drawing2D.HatchStyle.DottedGrid, null, GrapeCity.ActiveReports.Chart.Graphics.
PicturePutStyle.Stretched);

legend1.Border = new GrapeCity.ActiveReports.Chart.Border
(new GrapeCity.ActiveReports.Chart.Graphics.Line(System.Drawing.Color.Navy, 2),0,
System.Drawing.Color.Black);
legend1.DockArea = this.ChartControl1.ChartAreas[0];
legend1.LabelsFont = new GrapeCity.ActiveReports.Chart.FontInfo
(System.Drawing.Color.White, new System.Drawing.Font("Arial", 9F));
legend1.Header = lHeader;
legend1.MarginX = 5;
legend1.MarginY = 5;
this.ChartControl1.Legends.Add(legend1);

// Generate legend items
GrapeCity.ActiveReports.Chart.LegendItem legenditem = new
GrapeCity.ActiveReports.Chart.LegendItem();
GrapeCity.ActiveReports.Chart.LegendItem legenditem2 = new
GrapeCity.ActiveReports.Chart.LegendItem();

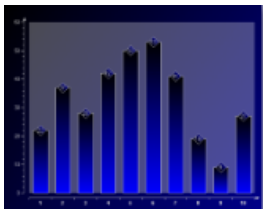
// Set properties of legend item
legenditem.Text = "High Range";
legenditem2.Text = "Low Range";
legenditem.Marker.Style = GrapeCity.ActiveReports.Chart.MarkerStyle.None;
legenditem2.Marker.Style = GrapeCity.ActiveReports.Chart.MarkerStyle.None;
legenditem.Border.Line.Style = GrapeCity.ActiveReports.Chart.Graphics.LineStyle.None;
legenditem2.Border.Line.Style = GrapeCity.ActiveReports.Chart.Graphics.LineStyle.None;
legenditem.Backdrop = new GrapeCity.ActiveReports.Chart.BackdropItem
(System.Drawing.Color.LightSteelBlue,
GrapeCity.ActiveReports.Chart.Graphics.AntiAliasMode.Auto);
legenditem2.Backdrop = new GrapeCity.ActiveReports.Chart.BackdropItem
(System.Drawing.Color.LightGray, GrapeCity.ActiveReports.Chart.Graphics.AntiAliasMode.Auto);

// Add to legend collection
legend1.LegendItems.Add(legenditem);
legend1.LegendItems.Add(legenditem2);

// set the legend property of the series to the legend you created.
this.ChartControl1.Series[0].Legend = legend1;
this.ChartControl1.Series[1].Legend = legend1;
this.ChartControl1.Series[2].Legend = legend1;
```

Markers

Use markers to show specific data series values in a chart. Markers are created by setting the Marker property of the series.



The following code demonstrates how to create a marker object at run time and assign it to the Marker property of the Series object. The results are shown in the image above.

To write code in Visual Basic.NET

Visual Basic code. Paste INSIDE the section Format event

```
' create the marker object
Dim marker1 As New GrapeCity.ActiveReports.Chart.Marker

' set the marker properties.
marker1.Backdrop = New Chart.Graphics.Backdrop(Chart.Graphics.GradientType.Horizontal, Color.Navy, Color.Black)
marker1.Line = New Chart.Graphics.Line(Color.White)
marker1.Label = New Chart.LabelInfo(New Chart.Graphics.Line(Color.Transparent, 0, Chart.Graphics.LineStyle.None),
 New Chart.Graphics.Backdrop(Chart.Graphics.BackdropStyle.Transparent, Color.White, Color.White,
 _ Chart.Graphics.GradientType.Vertical, System.Drawing.Drawing2D.HatchStyle.DottedGrid, Nothing,
 _Chart.Graphics.PicturePutStyle.Stretched),
 New Chart.FontInfo(Color.White, New Font("Arial", 8.0F)), "{Value}", Chart.Alignment.Center)
marker1.Size = 24
marker1.Style = Chart.MarkerStyle.Diamond

' assign the marker to the series Marker property
Me.ChartControl1.Series(0).Marker = marker1
```

To write code in C#

C# code. Paste INSIDE the section Format event

```
// create the marker object
GrapeCity.ActiveReports.Chart.Marker marker1 = new GrapeCity.ActiveReports.Chart.Marker();

// set the marker properties
marker1.Backdrop = new
GrapeCity.ActiveReports.Chart.Graphics.Backdrop(GrapeCity.ActiveReports.Chart.Graphics.GradientType.Horizontal,
 System.Drawing.Color.Navy, System.Drawing.Color.Black);
marker1.Line = new GrapeCity.ActiveReports.Chart.Graphics.Line(System.Drawing.Color.White);
marker1.Label = new GrapeCity.ActiveReports.Chart.LabelInfo(new GrapeCity.ActiveReports.Chart.Graphics.Line
(System.Drawing.Color.Transparent, 0, GrapeCity.ActiveReports.Chart.Graphics.LineStyle.None),
new
GrapeCity.ActiveReports.Chart.Graphics.Backdrop(GrapeCity.ActiveReports.Chart.Graphics.BackdropStyle.Transparent,
 System.Drawing.Color.White, System.Drawing.Color.White,
GrapeCity.ActiveReports.Chart.Graphics.GradientType.Vertical,
System.Drawing.Drawing2D.HatchStyle.DottedGrid, null,
GrapeCity.ActiveReports.Chart.Graphics.PicturePutStyle.Stretched),
 new GrapeCity.ActiveReports.Chart.FontInfo(System.Drawing.Color.White, new System.Drawing.Font("Arial", 8F)),
 "{Value}", GrapeCity.ActiveReports.Chart.Alignment.Center);
marker1.Size = 24;
marker1.Style = GrapeCity.ActiveReports.Chart.MarkerStyle.Diamond;

// assign the marker to the series Marker property
this.ChartControl1.Series[0].Marker = marker1;
```

Label Symbols

You can use Labels in Chart markers or legends.

By default, marker labels display Y value of data points, whereas legend labels display series name or data name.

Setting Strings

You can change a string (format string) displayed in a marker or legend label.

To change a string in a marker label

1. Display the Series collection editor of properties window.
2. Select the series that sets marker. (By default the first series (Series1) gets selected.
3. Expand the Properties property.
4. Expand the Marker property.
5. Expand the Label property.
6. Set the string to display in the Format property.

To change a string in a legend label

1. Display the Series collection editor of properties window.
2. Set the string you want to set in legend label using the **LegendText** property.

When the **LegendItemMode** property of Series is set to Series, series are displayed in legend. Data point will be displayed when set to Point. By default, in each graph the common setting will get displayed. For example, in case of bar chart it is series and in case of pie chart it is data point that gets displayed in legends.

Symbols

It is possible to easily display constant strings by simply performing the above mentioned settings. To display data, you need to embed the section (placeholder) that displays the value within format string at run time.

A placeholder is a particular symbol enclosed within brackets {}.

The following symbols can be used. The sections enclosed within {} are changed by values.

Value

Data Value（Y value）

Pct

Percentage within series

PPct

Percentage having 100% as sum of multiple series for data points.

Name

X value of data

Index

Index of data point

Total

Total number of series

PTotal

Total number of multiple series for data points

When displaying numeric value, it is possible to set format specifying string similar to System.String.Format method used when displaying numeric values. For example, when format string is {Value}, numeric value is displayed in default format but when ":" (colon) format specifying string is added after Value, it is possible to insert comma or specify decimal place digits.

For example, the format string for "inserting comma in numeric value and displaying 2 digits after decimal place" would be as follows.

{Value:#,##0.00}

Please refer to the technical information posted on Microsoft site for details on format specifying string after continued numeric value after comma.

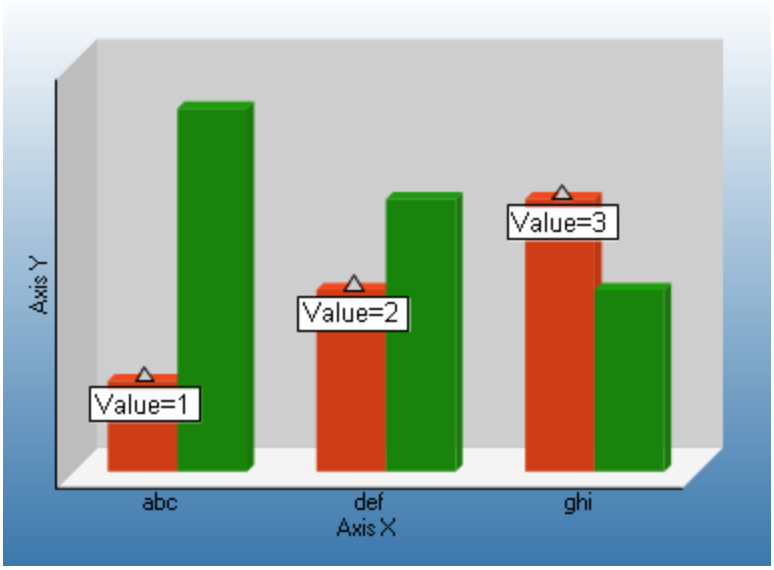
- [Format Function]
- [User defined numeric value format (Format function)]

Sample Image

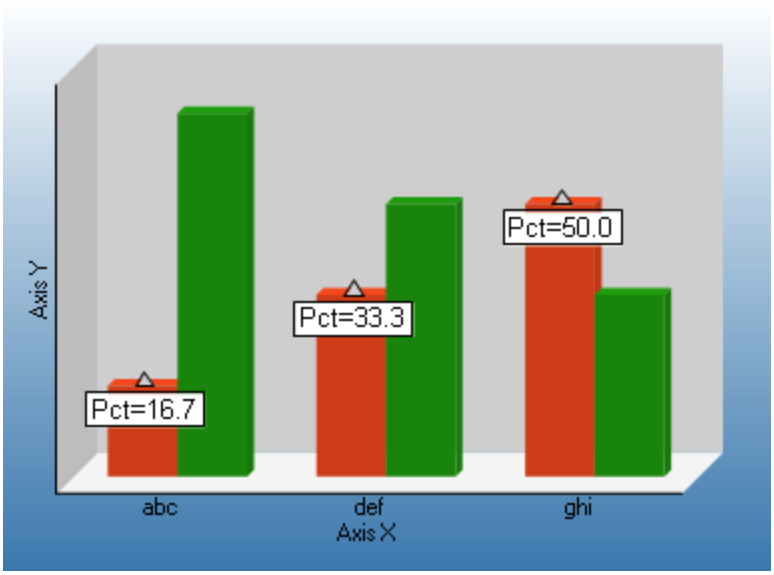
The following image displays a bar chart with the following values.

X Value	abc	def	ghi
Y value of series 1（Red）	1	2	3
Y value of series 2（green）	4	3	2

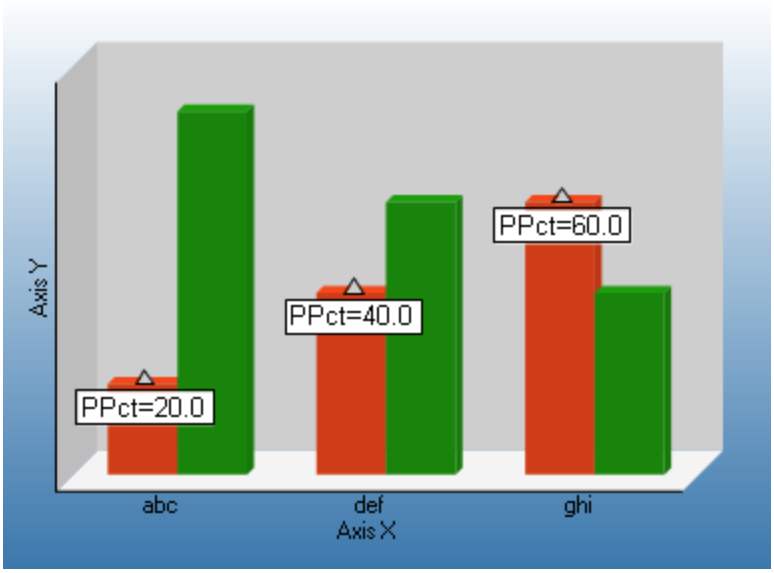
String displayed below the image is a string set in **Label.Format** property of marker.



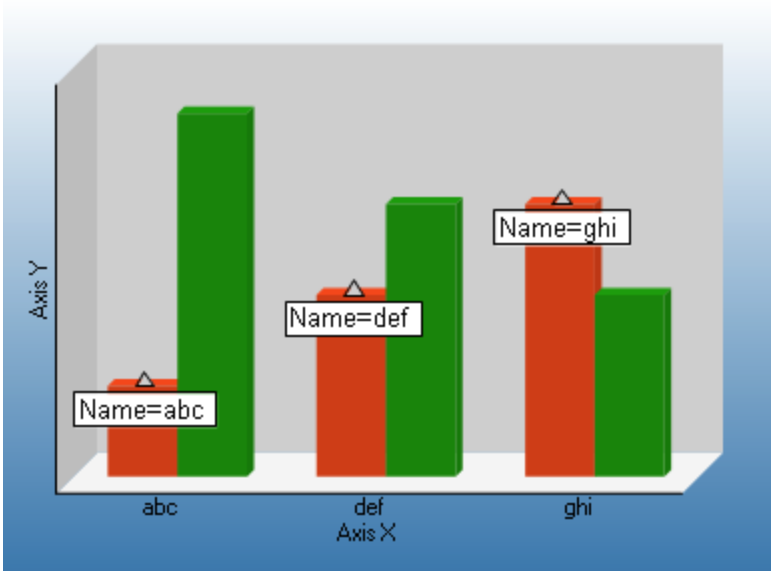
Value={Value}



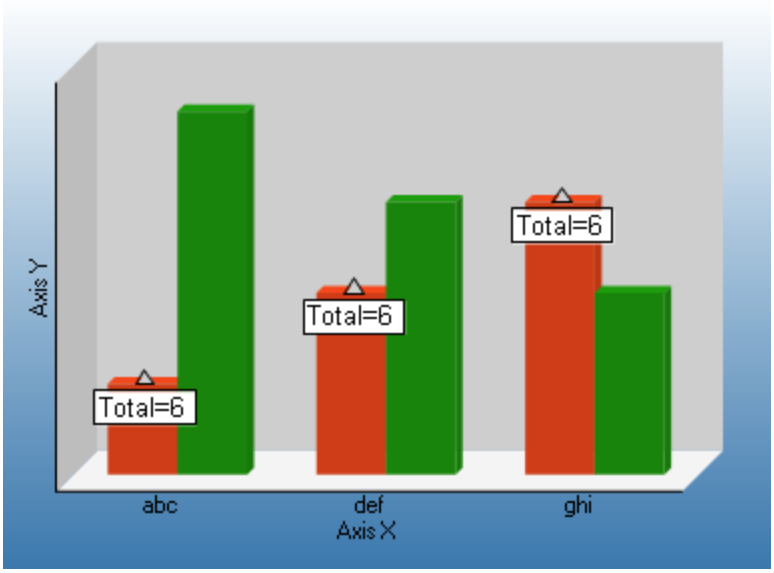
Pct={Pct:0.0}



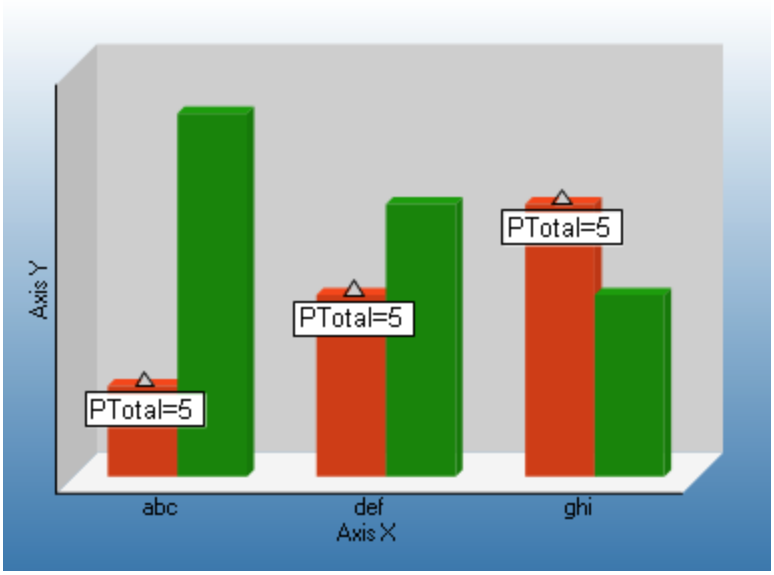
PPct={PPct:0.0}



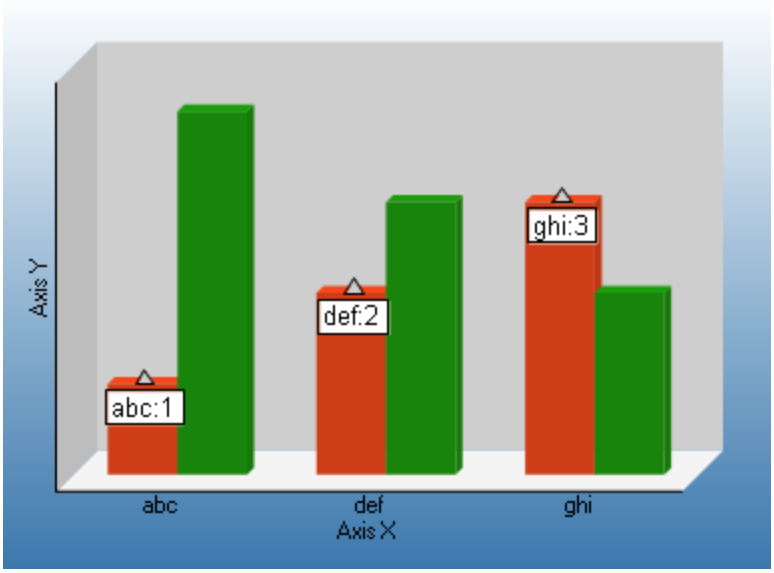
Name={Name}



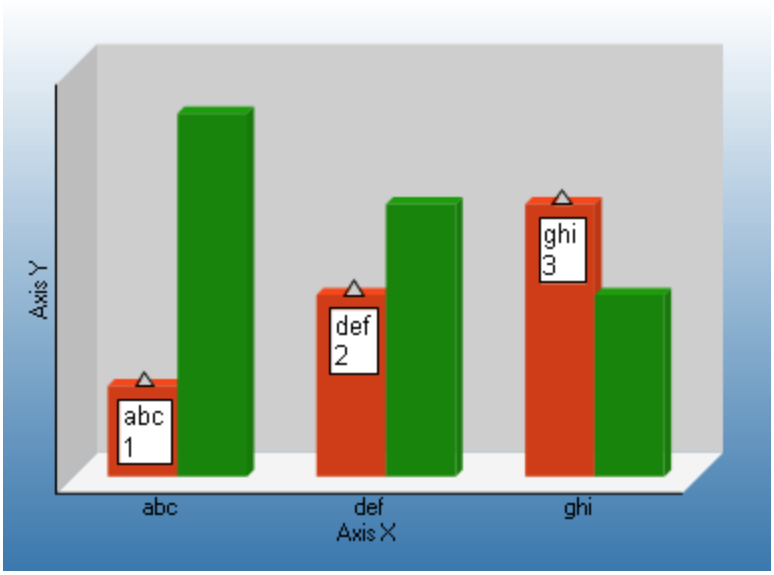
Total={Total}



PTotal={PTotal}



{Name}:{Value}



In case you wish to add a line break in between, as it is not possible to add a line break from Series collection editor, the following format string needs to be set at run time.

To write code in Visual Basic.NET

Visual Basic

```
Private Sub rptLabelSymbol4_ReportStart(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.ReportStart

Dim m As GrapeCity.ActiveReports.Chart.Marker
m = CType(Me.ChartControl1.Series(0).Properties("Marker"),
GrapeCity.ActiveReports.Chart.Marker)
m.Label.Format = "{Name}" & vbCrLf & "{Value}"
End Sub
```

To write code in C#

C#

```
private void NewActiveReport1_ReportStart(object sender, EventArgs e)
{
GrapeCity.ActiveReports.Chart.Marker m;
m = (GrapeCity.ActiveReports.Chart.Marker)
this.ChartControl1.Series[0].Properties["Marker"];
m.Label.Format = "{Name}\n{Value}";
}
```

Specific symbols

The specific symbols that are used according to the chart type.

Bubble Charts

Value

Y2 value

Value0

Y value

Value1

Y2value

Bubble XY Chart

Value0

X value

Value1

Y value

Value2

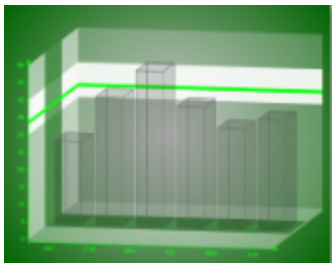
Y2 value

Candle or Hilo Chart

For a Candle chart or HiLo chart, symbols are not enabled for legend or marker labels.

Constant Lines and Stripes

The Chart control supports constant lines and stripes through the use of the **WallRanges** collection. It allows you to display horizontal or vertical lines or stripes in a chart to highlight certain areas. For example, you could draw a stripe in a chart to draw attention to a high level in the data or draw a line to show the average value of the data presented.



Note: The Chart control does not aggregate average values. Please aggregate the average values beforehand and then render the line.

Important properties

- **StartValue** Sets the start value on the primary axis for the wall range.

- **EndValue** Sets the end value on the primary axis for the wall range.
- **PrimaryAxis** Sets the axis on which the wall range appears.

The following code demonstrates how to create wall ranges, set their properties, and assign them to a chart area at run time. The results are shown in the image above.

To write code in Visual Basic.NET

Visual Basic code. Paste INSIDE the section Format event

```
' Create the WallRange objects
Dim wallRange1 As New GrapeCity.ActiveReports.Chart.WallRange
Dim wallRange2 As New GrapeCity.ActiveReports.Chart.WallRange
Dim wallRange3 As New GrapeCity.ActiveReports.Chart.WallRange

' Set WallRange property
With wallRange1
    .Backdrop = New GrapeCity.ActiveReports.Chart.Graphics.Backdrop(Color.White)
    .Border = New GrapeCity.ActiveReports.Chart.Border(New
GrapeCity.ActiveReports.Chart.Graphics.Line(Color.Transparent, 0,
GrapeCity.ActiveReports.Chart.Graphics.LineStyle.None), 0, Color.Black)
    .EndValue = 40
    .PrimaryAxis = ((CType(Me.ChartControl1.ChartAreas(0).Axes("AxisY"),
GrapeCity.ActiveReports.Chart.Axis)))
    .StartValue = 30
End With
With wallRange2
    .Backdrop = New GrapeCity.ActiveReports.Chart.Graphics.Backdrop(Color.Lime)
    .Border = New GrapeCity.ActiveReports.Chart.Border(New
GrapeCity.ActiveReports.Chart.Graphics.Line(Color.Transparent, 0,
GrapeCity.ActiveReports.Chart.Graphics.LineStyle.None), 0, Color.Black)
    .EndValue = 34
    .PrimaryAxis = ((CType(Me.ChartControl1.ChartAreas(0).Axes("AxisY"),
GrapeCity.ActiveReports.Chart.Axis)))
    .StartValue = 33
End With
With wallRange3
    .Backdrop = New GrapeCity.ActiveReports.Chart.Graphics.Backdrop(Color.DarkGreen,
CType(150, Byte))
    .Border = New GrapeCity.ActiveReports.Chart.Border(New
GrapeCity.ActiveReports.Chart.Graphics.Line(Color.Transparent, 0,
GrapeCity.ActiveReports.Chart.Graphics.LineStyle.None), 0, Color.Black)
    .EndValue = 40
    .PrimaryAxis = ((CType(Me.ChartControl1.ChartAreas(0).Axes("AxisZ"),
GrapeCity.ActiveReports.Chart.Axis)))
    .StartValue = 20
End With

' Add the WallRange to the chart area and set wall and Z axis properties to show lines.
```

```
With ChartControl1.ChartAreas(0)
    .WallRanges.AddRange(New GrapeCity.ActiveReports.Chart.WallRange() {wallRange1,
wallRange2, wallRange3})
    .WallXY.Backdrop.Alpha = 100
    .WallXZ.Backdrop.Alpha = 100
    .WallYZ.Backdrop.Alpha = 100
    .Axes(4).MajorTick.Step = 20
    .Axes(4).Max = 60
    .Axes(4).Min = 0
    .Axes(4).Visible = True
End With
```

To write code in C#

C# code. Paste INSIDE the section Format event

```
// Create the WallRange objects
GrapeCity.ActiveReports.Chart.WallRange wallRange1 = new
GrapeCity.ActiveReports.Chart.WallRange();
GrapeCity.ActiveReports.Chart.WallRange wallRange2 = new
GrapeCity.ActiveReports.Chart.WallRange();
GrapeCity.ActiveReports.Chart.WallRange wallRange3 = new
GrapeCity.ActiveReports.Chart.WallRange();

// Set WallRange property
wallRange1.Backdrop = new
GrapeCity.ActiveReports.Chart.Graphics.Backdrop(System.Drawing.Color.White);
wallRange1.Border = new GrapeCity.ActiveReports.Chart.Border(new
GrapeCity.ActiveReports.Chart.Graphics.Line
(System.Drawing.Color.Transparent, 0,
GrapeCity.ActiveReports.Chart.Graphics.LineStyle.None),
0, System.Drawing.Color.Black);
wallRange1.EndValue = 40;
wallRange1.PrimaryAxis =
(GrapeCity.ActiveReports.Chart.Axis)this.ChartControl1.ChartAreas[0].Axes["AxisY"];
wallRange1.StartValue = 30;
wallRange2.Backdrop = new
GrapeCity.ActiveReports.Chart.Graphics.Backdrop(System.Drawing.Color.Lime);
wallRange2.Border = new GrapeCity.ActiveReports.Chart.Border(new
GrapeCity.ActiveReports.Chart.Graphics.Line
(System.Drawing.Color.Transparent, 0,
GrapeCity.ActiveReports.Chart.Graphics.LineStyle.None),
0, System.Drawing.Color.Black);
wallRange2.EndValue = 34;
wallRange2.PrimaryAxis =
(GrapeCity.ActiveReports.Chart.Axis)this.ChartControl1.ChartAreas[0].Axes["AxisY"];
wallRange2.StartValue = 33;
wallRange3.Backdrop = new
```

```
GrapeCity.ActiveReports.Chart.Graphics.Backdrop(System.Drawing.Color.DarkGreen);
wallRange3.Border = new GrapeCity.ActiveReports.Chart.Border(new
GrapeCity.ActiveReports.Chart.Graphics.Line
(System.Drawing.Color.Transparent, 0,
GrapeCity.ActiveReports.Chart.Graphics.LineStyle.None),
0, System.Drawing.Color.Black);
wallRange3.EndValue = 40;
wallRange3.PrimaryAxis =
(GrapeCity.ActiveReports.Chart.Axis)this.ChartControl1.ChartAreas[0].Axes["AxisZ"];
wallRange3.StartValue = 20;

// Add the WallRange to the chart area and set wall and Z axis properties to show lines.
this.ChartControl1.ChartAreas[0].WallRanges.AddRange(
new GrapeCity.ActiveReports.Chart.WallRange[] {wallRange1,wallRange2,wallRange3});
this.ChartControl1.ChartAreas[0].WallXY.Backdrop.Alpha = 100;
this.ChartControl1.ChartAreas[0].WallXZ.Backdrop.Alpha = 100;
this.ChartControl1.ChartAreas[0].WallYZ.Backdrop.Alpha = 100;
this.ChartControl1.ChartAreas[0].Axes[4].MajorTick.Step = 20;
this.ChartControl1.ChartAreas[0].Axes[4].Max = 60;
this.ChartControl1.ChartAreas[0].Axes[4].Min = 0;
this.ChartControl1.ChartAreas[0].Axes[4].Visible = true;
```

Chart Axes and Walls

This section explains about the axis and walls of the Chart data region.

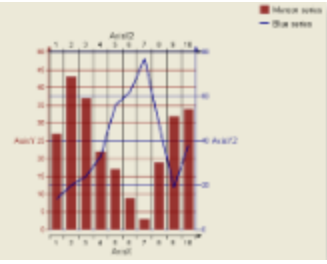
Standard Axes
: This section explains about standard axis properties or special characteristics.

Custom Axes
: This section explains about the method to create custom axis.

Gridlines and Tick Marks
: This section explains about the usage of grid lines and ticks.

Standard Axes

The Chart control provides the means to change axis settings at design time or at run time. Chart axes make it possible to view and understand the data plotted in a graph.



Axis Types

Most 2D charts contain a numerical axis (AxisY) and a categorical axis (AxisX). 3D charts include another numerical axis (AxisZ). These axes are accessible at run time from the ChartArea object and allow you to control the settings for each, including scaling, labels, and various formatting properties. For any of the scaling or labeling properties you set to show up at run time, you will need to set the **Visible** property of the axis to True.

Changing Axis Settings

Axis settings can be changed at design time by clicking on a Chart control and using the Properties Window or at run time in code from the chart's **ChartArea** object.

Scaling

For normal linear scaling on a numeric axis, set the **Max** and **Min** properties for the axis, which correspond to the numerical values in the chart's data series. Also, set the **Step** property of the MajorTick to show the major numerical unit values. The Step property controls where labels and tick marks are shown on the numerical axis.

To write code in Visual Basic.NET

Visual Basic code. Paste INSIDE the section Format event

```
With Me.ChartControl1.ChartAreas(0).Axes("AxisY")
    .Max = 100
    .Min = 0
    .MajorTick.Step = 10
End With
```

To write code in C#

C# code. Paste INSIDE the section Format event

```
this.ChartControl1.ChartAreas[0].Axes["AxisY"].Max = 100;
this.ChartControl1.ChartAreas[0].Axes["AxisY"].Min = 0;
this.ChartControl1.ChartAreas[0].Axes["AxisY"].MajorTick.Step = 10;
```

The Chart control also supports logarithmic scaling which allows you to show the vertical spacing between two points that corresponds to the percentage of change between those numbers. You can set your numeric axis to scale logarithmically by setting the **IsLogarithmic** property on the axis to True and setting the **Max** and **Min** properties of the axis.

Labeling

To show labels on an axis, you will need to specify the value for the **LabelsGap** property, set your **LabelsFont** properties, and set **LabelsVisible** to True. These properties can be set in the AxisBase Collection editor, which is accessed at design time by clicking the ellipsis button next to the **ChartAreas (Collection)** property, then the **Axes (Collection)** property of the ChartArea.

> **Tip**: Labels render first, and then the chart fills in the remaining area, so be sure to make the chart large enough if you use angled labels.

You can specify strings to be used for the labels instead of numerical values on an axis by using the Labels collection

property at design time or assigning a string array to the Labels property at run time. You can also specify whether you want your axis labels to appear on the outside or inside of the axis line using the **LabelsInside** property. By default, labels appear outside the axis line.

Format for numeric value axis label

You can use the **LabelFormat** property to set the label format of numeric value axis. The LabelFormat property can be referenced from the AxisBase collection editor.

For example, if you want to display a digit separator as shown in the image above, you should specify "{0:c}" in the **LabelFormat** property to output the "\1,213" value. When "{0:#,###円}" is specified, the output value is "1,213円". The LabelFormat property uses the same format as the ListControl.DataFormatString property of .NET Framework standard control.

> **Tip**: To set a specific tick interval using the LabelFormat property, the **SmartLabels** property should be set to False.

Secondary Axes

By default, a Chart object includes secondary X and Y axes (AxisX2 and AxisY2). At design time or run time, you can specify a secondary axis to plot data against by setting all of the appropriate properties for AxisX2 or AxisY2, including the **Visible** property.For example, if you want to use two axes to show the same data as it appears on two different scales, you can set the primary axis to show the actual data value scale, for example, and set the secondary axis to show a logarithmic scale.

To write code in Visual Basic.NET

Visual Basic code. Paste INSIDE the section Format event

```
' set properties for AxisY (primary axis)
With Me.ChartControl1.ChartAreas(0).Axes("AxisY")
   .Max = 25
   .Min = 0
   .MajorTick.Step = 5
End With

' set properties for AxisY2 (secondary Y axis)
With Me.ChartControl1.ChartAreas(0).Axes("AxisY2")
   .Max = 1000
   .Min = 0
   .MajorTick.Step = 200
' set the scaling for the secondary axis to logarithmic
   .AxisType = GrapeCity.ActiveReports.Chart.AxisType.Logarithmic
   .Visible = True
End With
```

To write code in C#

C# code. Paste INSIDE the section Format event

```
// set properties for AxisY (primary axis)
this.ChartControl1.ChartAreas[0].Axes["AxisY"].Max = 25;
this.ChartControl1.ChartAreas[0].Axes["AxisY"].Min = 0;
```

```
this.ChartControl1.ChartAreas[0].Axes["AxisY"].MajorTick.Step = 5;

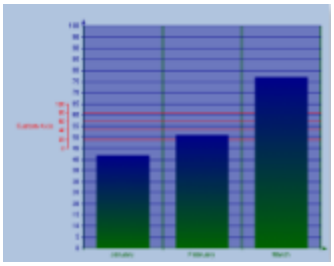
// set properties for AxisY2 (secondary Y axis)
this.ChartControl1.ChartAreas[0].Axes["AxisY2"].Max = 1000;
this.ChartControl1.ChartAreas[0].Axes["AxisY2"].Min = 0;
this.ChartControl1.ChartAreas[0].Axes["AxisY2"].MajorTick.Step = 200;

// set the scaling for the secondary axis to logarithmic
this.ChartControl1.ChartAreas[0].Axes["AxisY2"].AxisType =
GrapeCity.ActiveReports.Chart.AxisType.Logarithmic;
this.ChartControl1.ChartAreas[0].Axes["AxisY2"].Visible = true;
```

Custom Axes

The Chart control supports the creation of additional custom axes through the use of the chart's CustomAxes collection. Once a custom axis has been added to the collection, in addition to setting the normal axis properties, you will need to set the following properties:

- **Parent** - The Parent property allows you to choose the primary or secondary axis on which your custom axis resides.
- **PlacementLength** - The PlacementLength property allows you to set the length of the custom axis in proportion to the Min and Max property values you have already set for the parent axis.
- **PlacementLocation** - The PlacementLocation property allows you to set the starting location value for the custom axis to appear in relation to the parent axis.



The following code sample demonstrates creating a custom axis, adding it to the Axes collection for the ChartArea, and setting its properties.

To write code in Visual Basic.NET

Visual Basic code. Paste INSIDE the section Format event

```
' create the custom axis and add it to the ChartArea's Axes collection
Dim customAxisY As New GrapeCity.ActiveReports.Chart.CustomAxis
Me.ChartControl1.ChartAreas(0).Axes.Add(customAxisY)

' set the basic axis properties for customAxisY
customAxisY.LabelFont = New GrapeCity.ActiveReports.Chart.FontInfo(Color.Red, New
Font("Arial", 7.0!))
customAxisY.LabelsGap = 1
customAxisY.LabelsVisible = True
```

```
customAxisY.Line = New GrapeCity.ActiveReports.Chart.Graphics.Line(Color.Red, 1,
GrapeCity.ActiveReports.Chart.Graphics.LineStyle.Solid)
customAxisY.Max = 100
customAxisY.Min = 0
customAxisY.MaxDerived = False
customAxisY.Visible = True

' set major tick
customAxisY.MajorTick = New GrapeCity.ActiveReports.Chart.Tick(New
GrapeCity.ActiveReports.Chart.Graphics.Line(Color.Red, 1), New
GrapeCity.ActiveReports.Chart.Graphics.Line(Color.Red, 1), 20, 5, True)
customAxisY.MajorTick.Visible = True

' set minor tick
customAxisY.MinorTick.Visible = False
customAxisY.MinorTick.GridLine.Style = Chart.Graphics.LineStyle.None

' set custom axis property
customAxisY.Parent = ((CType(Me.ChartControl1.ChartAreas(0).Axes("AxisY"),
GrapeCity.ActiveReports.Chart.Axis)))
customAxisY.PlacementLength = 20
customAxisY.PlacementLocation = 45
```

To write code in C#

C# code. Paste INSIDE the section Format event

```
// create the custom axis and add it to the ChartArea's Axes collection
GrapeCity.ActiveReports.Chart.CustomAxis customAxisY = new
GrapeCity.ActiveReports.Chart.CustomAxis();
this.ChartControl1.ChartAreas[0].Axes.Add(customAxisY);

// set the basic axis properties for customAxisY
customAxisY.LabelFont = new GrapeCity.ActiveReports.Chart.FontInfo(Color.Red, new
Font("Arial", 7F, FontStyle.Regular, GraphicsUnit.Point, ((System.Byte)(0))));
customAxisY.LabelsGap = 1;
customAxisY.LabelsVisible = true;
customAxisY.Line = new GrapeCity.ActiveReports.Chart.Graphics.Line(Color.Red);
customAxisY.MajorTick = new GrapeCity.ActiveReports.Chart.Tick(new
GrapeCity.ActiveReports.Chart.Graphics.Line(Color.Red, 1), new
GrapeCity.ActiveReports.Chart.Graphics.Line(Color.Red, 1), 1, 2F, true);
customAxisY.MajorTick.GridLine = new
GrapeCity.ActiveReports.Chart.Graphics.Line(Color.Red, 1,
GrapeCity.ActiveReports.Chart.Graphics.LineStyle.Solid);
customAxisY.MajorTick.Visible = true;
customAxisY.Max = 5;
customAxisY.MaxDerived = false;
customAxisY.Min = 0;
```

```
customAxisY.Visible = true;

// set the special custom axis properties
customAxisY.Parent =
(GrapeCity.ActiveReports.Chart.Axis)this.ChartControl1.ChartAreas[0].Axes["AxisY"];
customAxisY.PlacementLength = 20;
customAxisY.PlacementLocation = 30;
```

Gridlines and Tick Marks

Gridlines and tick marks are generally used to help increase the readability of a chart.

Types

There are two kinds of gridlines and tick marks in the Chart control: major and minor. The properties for the major gridlines and tick marks are set on the MajorTick object of the particular axis and the properties for minor gridlines and ticks are set on the MinorTick object of the axis. The location for any labels shown for the axis are determined by the **Step** property of the MajorTick object.

Step and TickLength

For either the MajorTick or MinorTick objects, you can define where the tick marks and gridlines will appear by setting the Step property. The **TickLength** property allows you to set the region the tick mark will extend outside of the axis line.

> **Tip**: To set a specific tick interval using the **Step** property, you should set the **SmartLabels** property to False.

Visible

To make any defined major or minor tick marks to show up at design time or run time, the **Visible** property of the MajorTick or MinorTick object must be set to True. To show major or minor gridlines at design time or run time, the Visible property of the WallXY object as well as that of the MajorTick or MinorTick object must be set to True.

Report Info

In ActiveReports, the ReportInfo control allows you to quickly display page numbers, page counts, and report dates. The ReportInfo control is a text box with a selection of preset FormatString options. You can set page counts to count the pages for the entire report, or for a specified group.

You can customize the preset values by editing the string after you select it. For example, if you want to display the total

number of pages in the ReportHeader section, you can enter a value like:

```
Total of {PageCount} pages.
```

> ⚠ **Caution**: With large reports using the **CacheToDisk** property, placing page counts in header sections may have an adverse effect on memory as well as rendering speed. Since the rendering of the header is delayed until ActiveReports determines the page count of the following sections, CacheToDisk is unable to perform any optimization. For more information on this concept, see Optimizing Section Reports.

For more information on creating formatting strings, see the Date, Time, and Number Formatting topic.

Important Properties

Property	Description
FormatString	Gets or sets the string used to display formatted page numbering or report date and time values in the control.
Style	Gets or sets the style string for the control. This property reflects any settings you choose in the Font and ForeColor properties.
SummaryGroup	Gets or sets the name of the GroupHeader section that is used to reset the number of pages when displaying group page numbering.

Displaying page numbers and report dates

1. From the toolbox, drag the **ReportInfo** control to the desired location on the report.
2. With the ReportInfo control selected in the Properties window, drop down the **FormatString** property and select the preset value that best suits your needs.

Displaying group level page counts

1. From the toolbox, add the ReportInfo control to the GroupHeader or GroupFooter section of a report and set the **FormatString** property to a value that includes PageCount.
2. With the ReportInfo control still selected, in the Properties window, drop down the **SummaryGroup** property and select the group for which you want to display a page count.

ReportInfo Dialog

With the control selected on the report, in the Commands section at the bottom of the Properties window, you can click the **Property dialog** command to open the dialog.

General

Name: Enter a name for the control that is unique within the report. This name is displayed in the Document Outline and in XML exports. You can only use underscore (_) as a special character in the Name field. Other special characters such as period (.), space ( ), forward slash (/), back slash (\), exclamation (!), and hyphen (-) are not supported.

Tag: Enter a string that you want to persist with the control. If you access this property in code, it is an object, but in the Properties window or Property dialog, it is a string.

Visible: Clear this check box to hide the control.

DataField: Select a field name from the data source to which to bind the control.

Appearance

Background Color: Select a color to use for the background of the control.

Font

Name: Select a font family name or a theme font.

Size: Choose the size in points for the font.

Style: Choose **Normal** or **Italic**.

Weight: Choose from **Normal** or **Bold**.

Color: Choose a color to use for the text.

Decoration: Select check boxes for **Underline** and **Strikeout**.

GDI Charset: Enter a value to indicate the GDI character set to use. For a list of valid values, see MSDN Font.GDICharSet Property.

GDI Vertical: Select this checkbox to indicate that the font is derived from a GDI vertical font.

Format

Format string: Select a formatted page numbering or report date and time value to display in the control. You may also type in this box to change or add text to display along with the formatted date and page number values.

Multiline: Select this check box to allow text to render on multiple lines within the control.

ReportInfo height

Can increase to accommodate contents: Select this check box to set CanGrow to True.

Can decrease to accommodate contents: Select this check box to set CanShrink to True.

Text direction

RightToLeft: Select this check box to reverse the text direction.

Alignment

Vertical alignment: Choose **Top**, **Middle**, or **Bottom**.

Horizontal alignment: Choose **Left**, **Center**, **Right**, or **Justify**.

Wrap mode: Choose **NoWrap**, **WordWrap**, or **CharWrap** to select whether to wrap words or characters to the next line.

Summary

SummaryGroup: Select a GroupHeader section in the report to display the number of pages in each group when using the PageCount.

SummaryRunning: Select None, Group, or All to display a summarized value.

Cross Section Controls

In section reports, you can use the CrossSectionLine and CrossSectionBox report controls to display a frame,

borders, and vertical lines that run from a header section through its related footer section, spanning all details that come between. You can specify line appearance using properties on the report controls, and even round the corners of the CrossSectionBox.

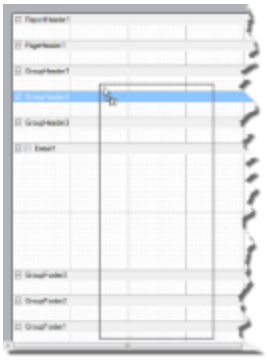
You can only place the cross section controls in header sections in the designer. They automatically span intervening sections to end in the related footer section. (You can also place them in footer sections, but they automatically associate themselves with the related header section in the Report Explorer.)

CrossSectionLine

The CrossSectionLine control draws a vertical line from a header section to the corresponding footer section. At run time, this vertical line stretches through any intervening sections. You can change the appearance of CrossSectionLine by changing the following properties.

Property	Description
LineColor	Allows you to get or set color of the line.
LineStyle	Allows you to select the line style from solid, dash, dotted, or double.
LineWeight	Allows you to specify the thickness of the line.

CrossSectionBox



The CrossSectionBox control draws a rectangle from a header section to its corresponding footer section. To change the appearance of the rectangle, you can use the following properties in addition to the ones mentioned above.

Property	Description
Radius	Sets the radius of each corner for the RoundRect shape type. You can select Default, TopLeft, TopRight, BottomLeft or BottomRight. Selecting Default sets the radius of all the corners of the CrossSectionBox control to a specified percentage. Default value = 0 (point).

BackColor	Sets the back color.

Note: At run time, the **BackColor** property renders first and the **LineColor** property renders last.

Caution:

- The CrossSectionBox and CrossSectionLine controls do not render properly in multi-column reports, that is, those in which a GroupHeader section has the **ColumnLayout** property set to True.
- The CrossSection control (CrossSectionBox and CrossSectionLine) is drawn across multiple sections. Therefore, it is not possible to change its appearance etc. in the event of a section. The appearance of the CrossSection control can be dynamically changed only in the ReportStart event.

CrossSectionLine Dialog

With the control selected on the report, in the Commands section at the bottom of the Properties window, you can click the **Property dialog** command to open the dialog.

General

Name: Enter a name for the control that is unique within the report. This name is displayed in the Document Outline and in XML exports. You can only use underscore (_) as a special character in the Name field. Other special characters such as period (.), space ( ), forward slash (/), back slash (\), exclamation (!), and hyphen (-) are not supported.

Tag: Enter a string that you want to persist with the control. If you access this property in code, it is an object, but in the Properties window or Property dialog, it is a string.

Visible: Clear this check box to hide the control.

Appearance

Line style: Select a line style to use for the control. You can set it to Transparent, Solid, Dash, Dot, DashDot, DashDotDot, or Double.

Line weight: Enter the width for the line.

Line color: Select a color to use for the line.

CrossSectionBox Dialog

With the control selected on the report, in the Commands section at the bottom of the Properties window, you can click the **Property dialog** command to open the dialog.

General

Name: Enter a name for the control that is unique within the report. This name is displayed in the Document Outline and in XML exports. You can only use underscore (_) as a special character in the Name field. Other special characters such as period (.), space ( ), forward slash (/), back slash (\), exclamation (!), and hyphen (-) are not supported.

Tag: Enter a string that you want to persist with the control. If you access this property in code, it is an object, but in the Properties window or Property dialog, it is a string.

Visible: Clear this check box to hide the control.

Appearance

Line style: Select a line style to use for the border line. You can set it to Transparent, Solid, Dash, Dot, DashDot,

DashDotDot, or Double.

Line weight: Enter the width for the border line.

Line color: Select a color to use for the border line.

Background color: Select a color to use for the background of the picture control.

CloseBorder: Select the checkbox to close the borders.

> **Note:** The **CloseBorder** property is only available for CrossSectionBox control placed in Section report's Group Header and Group Footer.

Rounded Rectangle: Specify the radius for each corner of the CrossSectionBox individually. Drag the handlers ◆ available at each corner of the CrossSectionBox to set the value of the radius at each corner.

> **Note:** To enable specific corners, check the CheckBox available near each corner of the CrossSectionBox control.

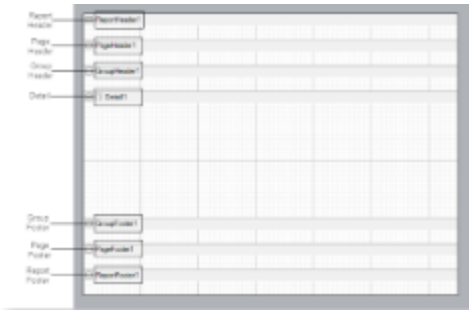
- **Use the same radius on specified corners**: Select this option to apply the same radius to all selected corners of the CrossSectionBox.
- **Use different radius on specified corners:** Select this option to apply a different radius to each selected corner of the CrossSectionBox.

Section Report Structure

By default, a section report is composed of three banded sections: a PageHeader, a Detail section, and a PageFooter. You can right-click the report and select **Insert** and choose other section pairs to add: ReportHeader and Footer, or GroupHeader and Footer.

All sections except the detail section come in pairs, above and below the detail section. You can hide any section that you are not using by setting the **Visible** property of the section to False.

ActiveReports defines the following section types:



Report Header

A report can have one report header section that prints at the beginning of the report. You generally use this section to print a report title, a summary table, a chart or any information that only needs to appear once at the report's start. This section has a **NewPage ('NewPage Property' in the on-line documentation)** property that you can use to add a new page before or after it renders.

The **Report Header** does not appear on a section report by default. In order to add this section, right-click the report and select Insert > Report Header/Footer to add a Report Header and Footer pair.

Page Header

A report can have one page header section that prints at the top of each page. Unless the page contains a report header section, the page header is the first section that prints on the page. You can use the page header to print column headers, page numbers, a page title, or any information that needs to appear at the top of each page.

Group Header

A report can include single or nested groups, with each group having its own header and footer sections. You can insert and print the header section immediately before the detail section. For more information on grouping, see Add Groups.

In Columnar Reports, you can use ColumnGroupKeepTogether, and select whether to start a NewColumn before or after a group.

You can also specify whether to print a NewPage before or after the section, and have the section print on every page until the group details complete with the RepeatStyle property. The UnderlayNext property allows you to show group header information inside the group details, so long as you keep the BackColor property of the Detail section set to Transparent.

See **GroupHeader ('GroupHeader Class' in the on-line documentation)** for further information on properties.

Detail

A report has one detail section. The detail section is the body of the report and one instance of the section is created for each record in the report. You can set the CanShrink property to True to eliminate white space after controls, and you can set up Columnar Reports using ColumnCount, ColumnDirection, ColumnSpacing and NewColumn properties.

The KeepTogether property attempts to keep the section together on a single page, and the **RepeatToFill ('RepeatToFill Property' in the on-line documentation)** property allows you to fill each page with the same number of formatted rows, regardless of whether there is enough data to fill them. This is especially useful for reports such as invoices in which you want consistent formatting like lines or green bars or back colors to fill each page down to the Footer section at the bottom.

See **Detail ('Detail Class' in the on-line documentation)** for further information on properties.

> **Note:** You cannot use the **RepeatToFill** property if you are using the PageBreak or SubReport control in the Detail section, or if you have set the NewPage or NewColumn property to any value other than None. When you use this property in a report where two groups are present, the ReportFooter section prints on the next page. This property processes correctly only with single grouping.

Group Footer

A report can include single or nested groups, with each group having its own header and footer sections. You can insert and print the footer section immediately after the detail section. For more information on grouping, see Add Groups.

Page Footer

A report can have one page footer section that prints at the bottom of each page. You can use the page footer to print page totals, page numbers, or any other information that needs to appear at the bottom of each page.

Report Footer

A report can have one report footer section that prints at the end of the report. Use this section to print a summary of the report, grand totals, or any information that needs to print once at the end of the report.

The **Report Footer** does not appear on a section report by default. In order to add this section, right-click the report and

select **Insert** > **Report Header/Footer** to add a Report Header and Footer pair.

> **Note:** If the report contains a Page Footer on the last page, the Report Footer appears above the Page Footer.

Section Report Events

Section reports use events to allow you to control report behavior.

Single-Occurrence Events

The following events are all of the events that are raised only once during a Section report's processing. These events are raised at the beginning or at the end of the report processing cycle.

Events raised once

ReportStart

Use this event to initialize any objects or variables needed while running a report. This event is also used to set any Subreport control objects to a new instance of the report assigned to the Subreport control.

> **Caution:** Be sure to add dynamic items to the report before this event finishes.

DataInitialize

This event is raised after ReportStart. Use it to add custom fields to the report's Fields collection. Custom fields can be added to a bound report (one that uses a Data Control to connect and retrieve records) or an unbound report (one that does not depend on a data control to get its records). In a bound report the dataset is opened and the dataset fields are added to the custom fields collection, then the DataInitialize event is raised so new custom fields can be added. The DataInitialize event can also be used to make adjustments to the DataSource or to set up database connectivity.

ReportEnd

This event is raised after the report finishes processing. Use this event to close or free any objects that you were using while running a report in unbound mode, or to display information or messages to the end user. This event can also be used to export reports.

Multiple-Occurrence Events

The following events are raised multiple times during a Section report's processing.

Events raised more than once

FetchData

This event is raised every time a new record is processed. The FetchData has an EOF parameter indicating whether the FetchData event should be raised. This parameter is not the same as the Recordset's EOF property and is defaulted to True. When working with bound reports (reports using a DataControl), the EOF parameter is automatically set by the

report; however, when working with unbound reports this parameter needs to be controlled manually.

Use the FetchData event with unbound reports to set the values of custom fields that were added in the DataInitialize event or with bound reports to perform special functions, such as combining fields together or performing calculations. The FetchData event should not have any references to controls on the report.

If you need to use a value from a Dataset with a control in the Detail section, set a variable in the FetchData event and use the variable in the section's Format event to set the value for the control. Please note that this method of setting a variable in the FetchData event and using it to set a control's value is only supported in the Detail_Format event.

Also use the FetchData event to increment counters when working with arrays or collections.

PageStart

This event fires before a page is rendered. Use this event to initialize any variables needed for each page when running an unbound report.

PageEnd

This event is raised after each page in the report is rendered. Use this event to update any variables needed for each page when running an unbound report.

When Bound and Unbound Data Values Are Set

1. The Fields collection is populated from the dataset that is bound to the report after the DataInitialize event is raised. (In an unbound report, the Fields collection values are not set to anything at this point.)
2. The FetchData event is raised, giving the user a chance to modify the Fields collection.
3. Any fields that are bound have the values transferred over.
4. The Format event is raised.

Events that Occur for Each Instance of Each Section

In a Section report, regardless of the type or content of the various sections, there are three events for each section: **Format**, **BeforePrint** and **AfterPrint**.

Section Events

Because there are many possible report designs, the event-raising sequence is dynamic in order to accommodate individual report demands. The only guaranteed sequence is that a section's Format event is raised before the BeforePrint event, which in turn occurs before the AfterPrint event but not necessarily all together. Reports should not be designed to rely on these events being raised in immediate succession.

> ⚠ **Important**: Never reference the report's Fields collection in these section events. Only reference the Fields collection in the**DataInitialize** and **FetchData** events.

Format

ActiveReports raises this event after the data is loaded and bound to the controls contained in a section, but before the section is rendered to a page.

The Format event is the only event in which you can change the section's height. Use this section to set or change the

properties of any controls or the section itself.

Also use the Format event to pass information, such as an SQL String, to a Subreport.

If the CanGrow or CanShrink property is True for the section or any control within the section, all of the growing and shrinking takes place in the Format event. Because of this, you cannot obtain information about a control or section's height in this event.

Because a section's height is unknown until the Format event finishes, it is possible for a section's Format event to be raised while the report is on a page to which the section is not rendered. For example, the Detail Format event is raised but the section is too large to fit on the page. This causes the PageFooter events and the PageEnd event to be raised on the current page, and the PageStart, any other Header events, and possibly the FetchData event to be raised before the section is rendered to the canvas on the next page.

BeforePrint

ActiveReports raises this event before the section is rendered to the page.

The growing and shrinking of the section and its controls have already taken place. Therefore, you can use this event to get an accurate height of the section and its controls. You can modify values and resize controls in the BeforePrint event, but you cannot modify the height of the section itself.

Also use this event to do page-specific formatting since the report knows which page the section will be rendered to when this event is raised. Once this event has finished, the section cannot be changed in any way because the section is rendered to the canvas immediately after this event.

> **Note:** If a section contains the SubReport control that occupies more than one page, the SubReport gets split into smaller parts at rendering. In this case, you can use the BeforePrint event - it will fire multiple times to get the height of each part of the rendered SubReport.

AfterPrint

ActiveReports raises this event after the section is rendered to the page.

Although AfterPrint was an important event prior to ActiveReports Version 1 Service Pack 3, it is rarely used in any of the newer builds of ActiveReports. This event is still useful, however, if you want to draw on the page after text has already been rendered to it.

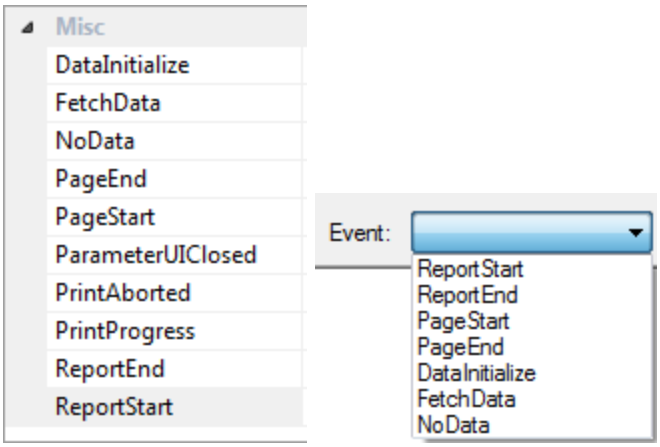
Event Sequence

Multi-threaded, single-pass processing enables Section reports to surpass other reports in processing and output generation speed. ActiveReports processes and renders each page as soon as the page is ready. If a page has unknown data elements or its layout is not final, it places the page in cache until the data is available.

Sequence of Events

Summary fields and KeepTogether constraints are two reasons why a page might not render immediately. The summary field is not complete until all the data needed for calculation is read from the data source. When a summary field such as a grand total is placed ahead of its completion level, such as in the report header, the report header and all following sections are delayed until all of the data is read.

There are ten report events in the code behind a Section report, or seven in a ActiveReport script.



Because there are so many ways in which you can customize your reports, not all reports execute in the same way. However, when you run a report, this is generally what happens:

1. ActiveReports raises the **ReportStart** event. The report validates any changes made to the report structure in ReportStart. In some cases, data source properties raise the **DataInitialize** event.
2. Printer settings are applied. If none are specified, the local machine's default printer settings are used.
3. If the **DataInitialize** event was not already raised, ActiveReports raises it and opens the data source.
4. If the data source contains Parameters with unset values and the **ShowParameterUI** property is set to **True**, ActiveReports displays a parameters dialog to request values from the user.
5. Closing the dialog raises the **ParameterUIClosed** event. If the report is a subreport that requires parameters, ActiveReports binds the subreport parameters to any fields in the parent report.
6. ActiveReports raises the **FetchData** event.
7. If there is no data, the **NoData** event is raised.
8. The **PageStart** event raises, and then raises again after each **PageEnd** event until the final page.
9. Group sections are bound and sections begin rendering on pages.
10. ActiveReports raises Section Events to process sections in (roughly) the following order:
    - Report header
    - Page header
    - Group header
    - Detail
    - Group footer
    - Page footer
    - Report footer
11. After each event, ActiveReports checks the **Cancel** flag to ensure that it should continue.
12. Other events may raise, depending on the report logic.
13. The **PageEnd** event raises after each page becomes full, and the **PageStart** raises if the report has not finished.
14. Finally, ActiveReports raises the **ReportEnd** event.

Events that May Occur

These events occur in response to user actions, or when there is no data for a report.

Other Events

DataSourceChanged

This event occurs if the report's data source is changed. This is mainly useful with the end-user designer control.

NoData

This event occurs if the report's data source returns no records.

ParameterUIClosed

This event occurs when the user closes the parameter dialog.

PrintAborted

This event occurs when the user cancels a print job.

PrintProgress

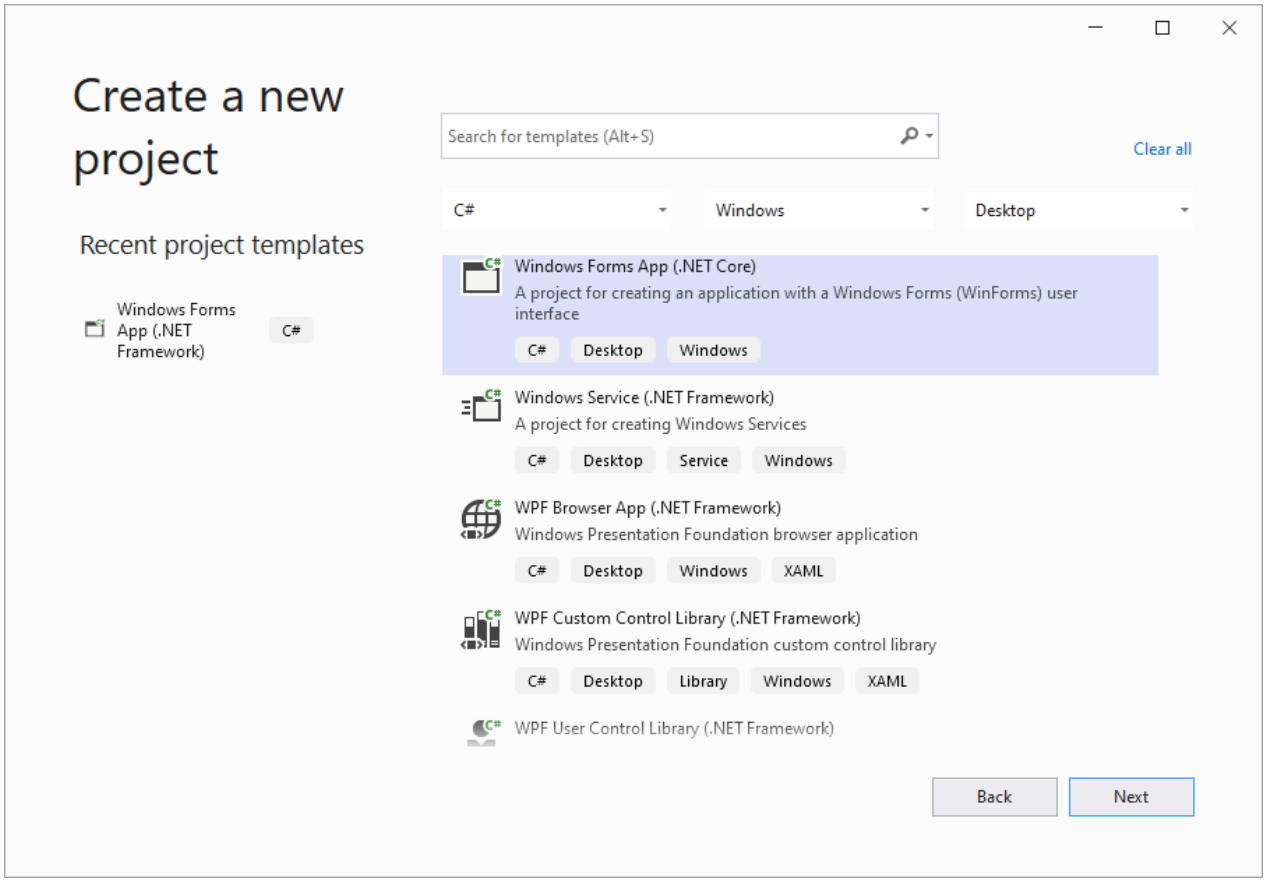
This event occurs once for each page while the report document is printing.

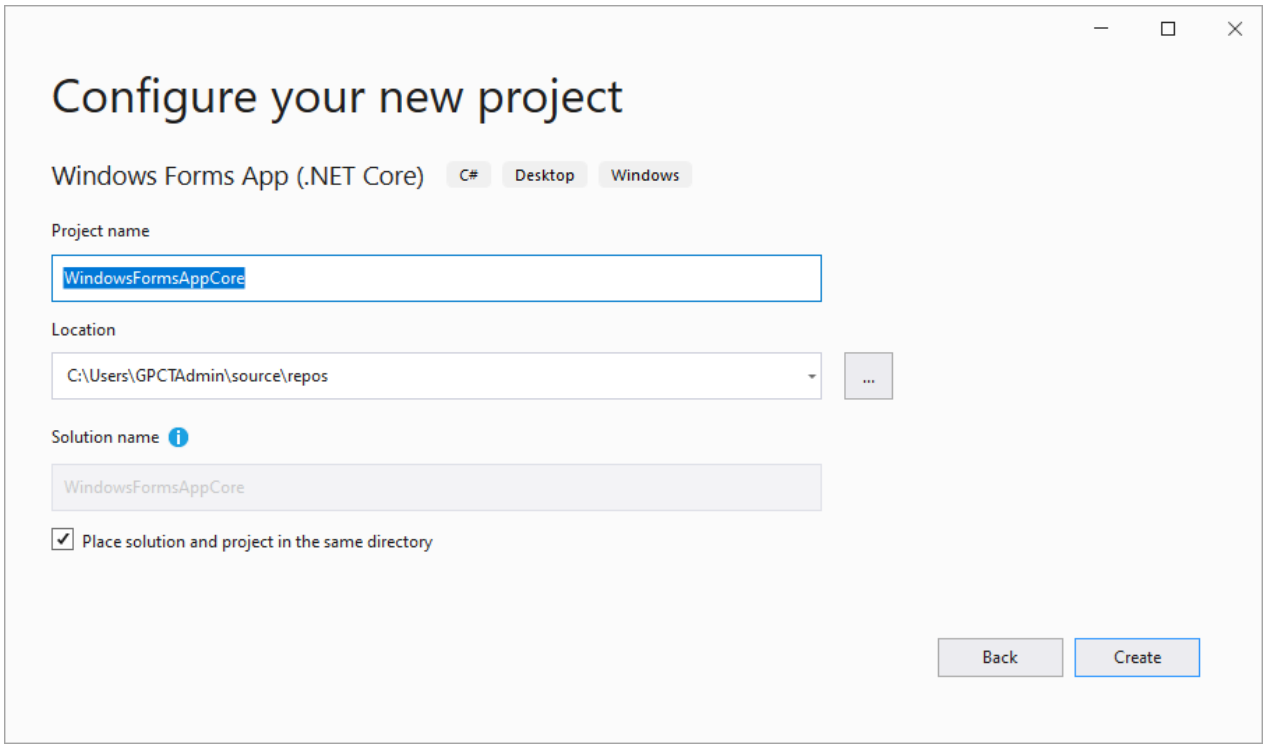
Designing Code-based Section Reports in .NET Core

The existing limitation in .NET Core 3.1 does not allow the ActiveReports Integrated Designer to be used for designing Code-based Section Reports in WinForms applications in Visual Studio.
As workaround, you need to use Visual Studio's option to link report files from .NET Core project in the .NET Framework project, and use the .NET Framework WinForms Designer.

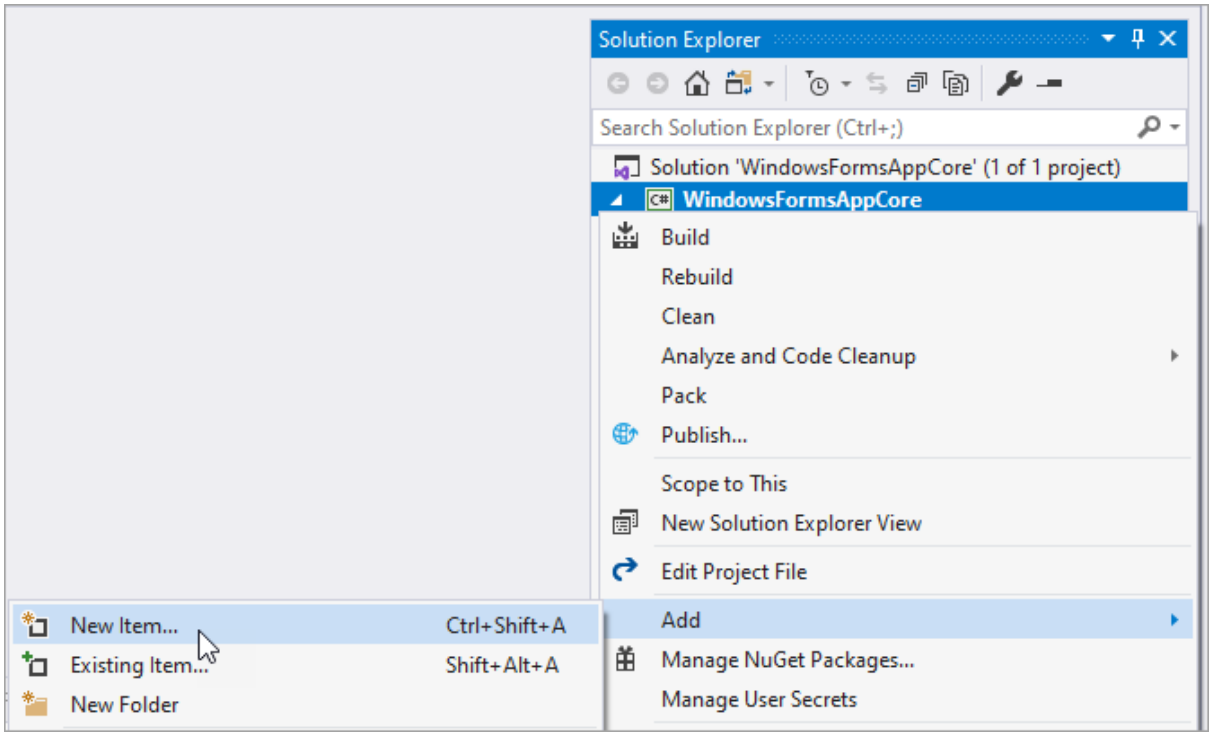
The steps to enable design-time report creation in .NET Core project for Code-Based Section reports are as follows:

1. **Create a new Windows Forms .NET Core project.**

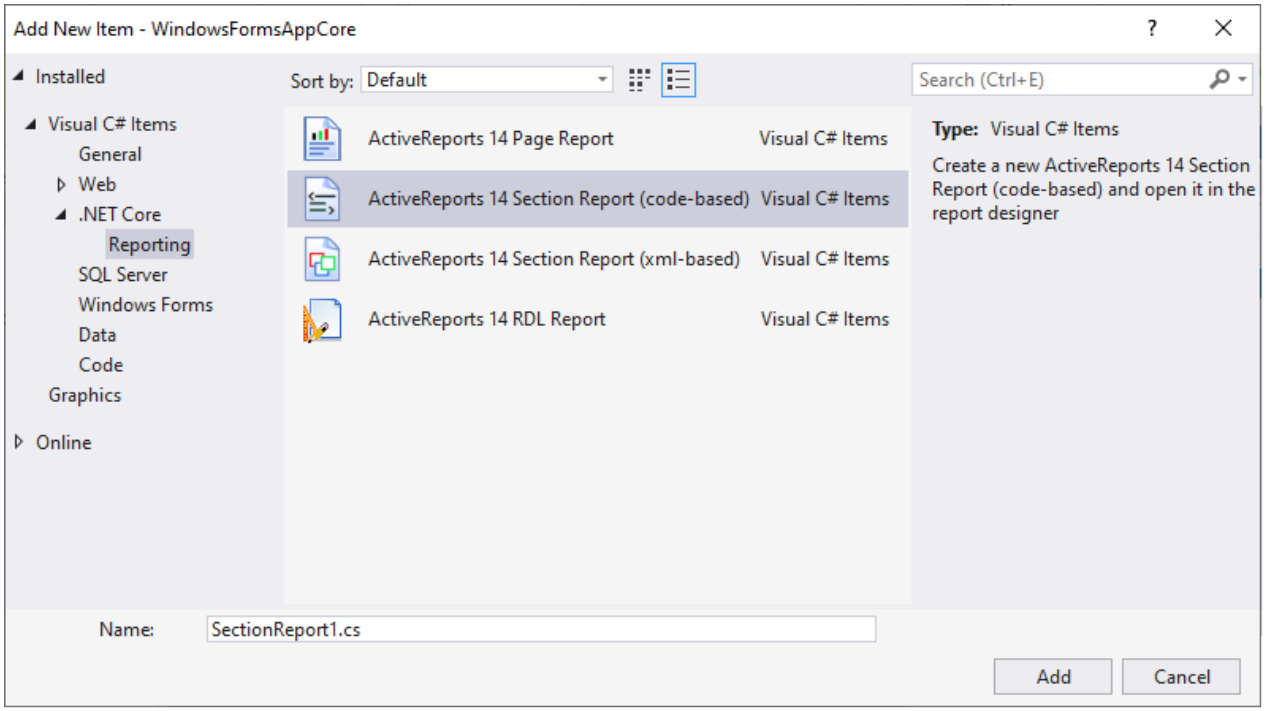




2. Add the code-based section report item to the .NET Core project. To do so, right-click the project and go to **Add > New Item**.

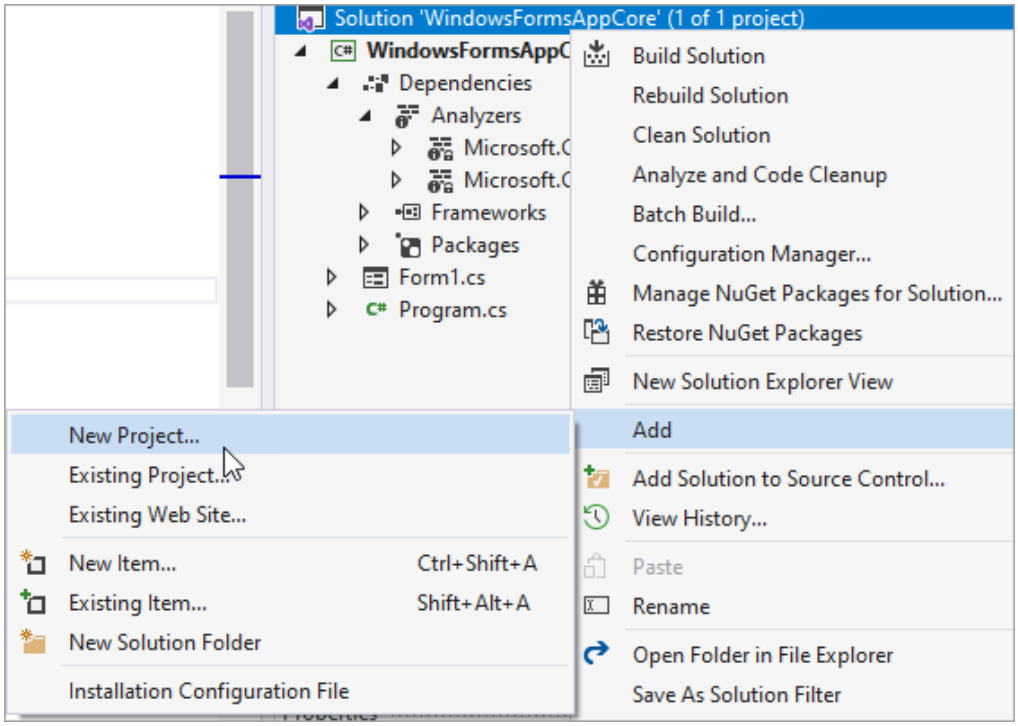


3. Select **ActiveReports 14 Section Report (code-based)** report item. All the required dependencies will be installed automatically.

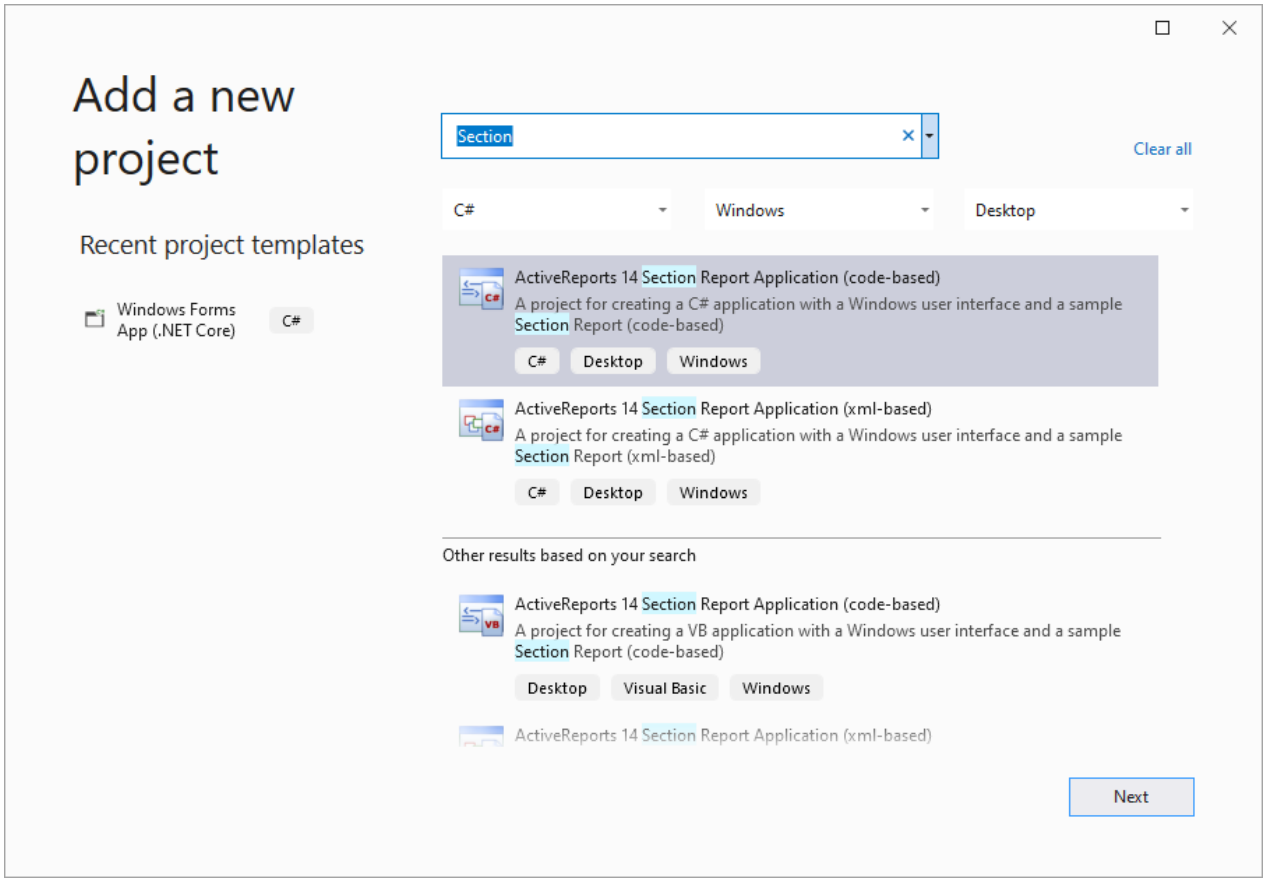


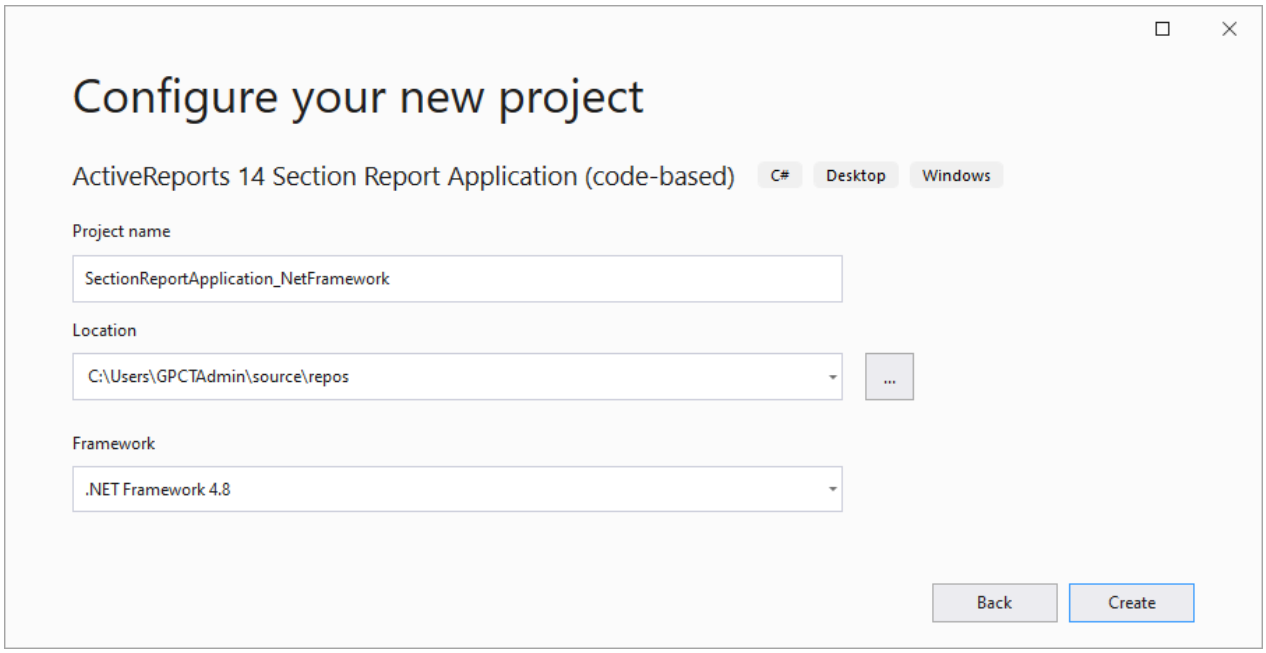
4. **Add a new .NET Framework project using built-in Section Report (code-based) template.**

    1. In the Solution Explorer, right-click the solution node and go to **Add** > **New Project**.

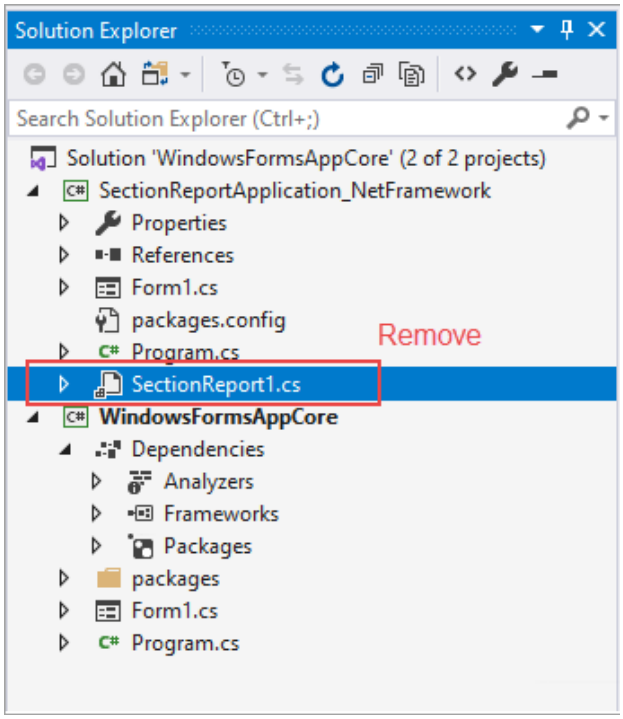


2. Select **ActiveReports 14 Section Report Application (code-based)** template and configure the project for the target .NET Framework version.



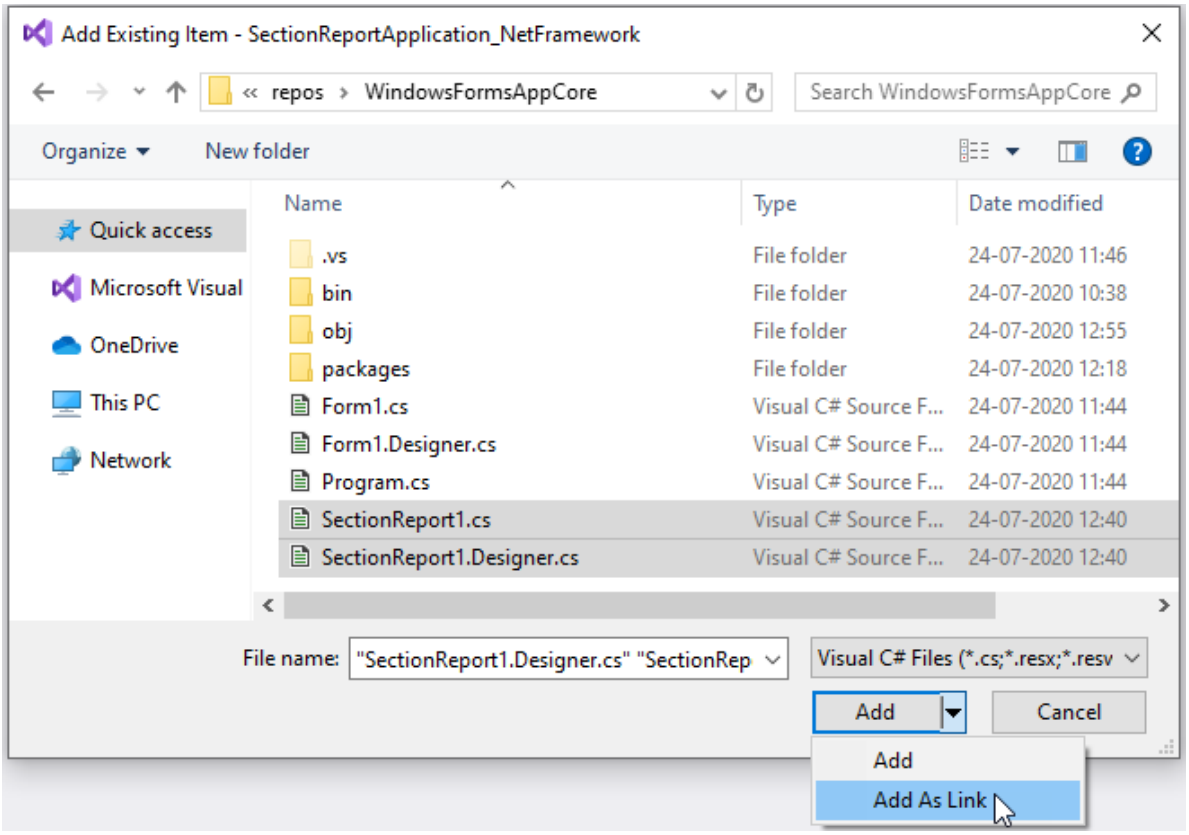


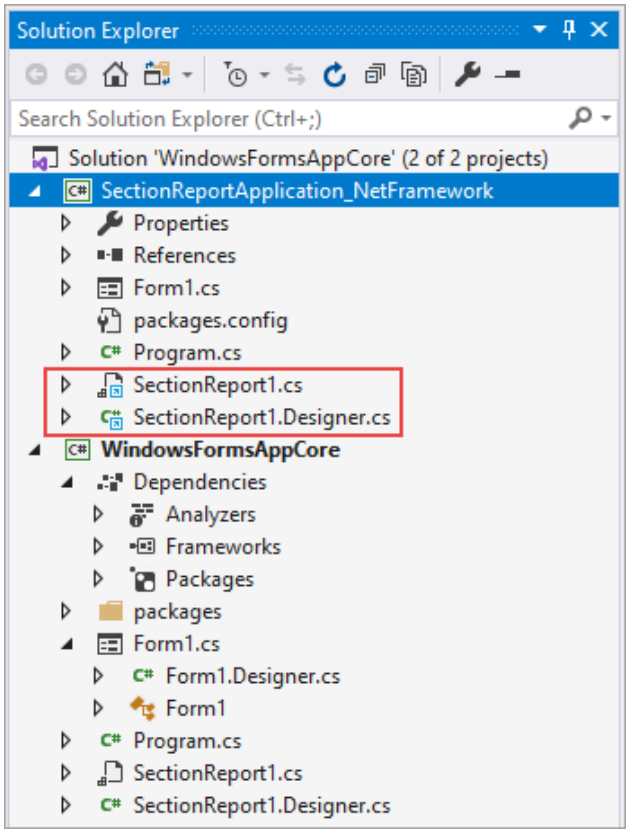
5. Remove the code-based report (SectionReport1.cs) from the .NET Framework project, which was added automatically in the previous step.



6. **Add report from the .NET Core project to the .NET Framework project as link.**

    1. In the Solution Explorer, right-click the .NET Framework project, go to **Add** > **Add Existing Item**.
    2. Navigate to the .NET Core project and select report files: 'SectionReport1.cs', 'SectionReport.Designer1.cs', and 'SectionReport1.resx'.
    3. Select **Add As Link** option to add the report files as link.





7. **Update the .csproj file of the .NET Framework project.**

   You need to add the dependency information in the .csproj file of the .NET Framework project. The instruction to do so are as follows:

   1. In the Solution Explorer, right, right-click the .NET Framework project node and select **Edit Project File**.
   2. In the **.csproj** file that opens, find the line

      ```
      <Compile Include="..\WindowsFormsAppCore\SectionReport1.Designer.cs">
      ```

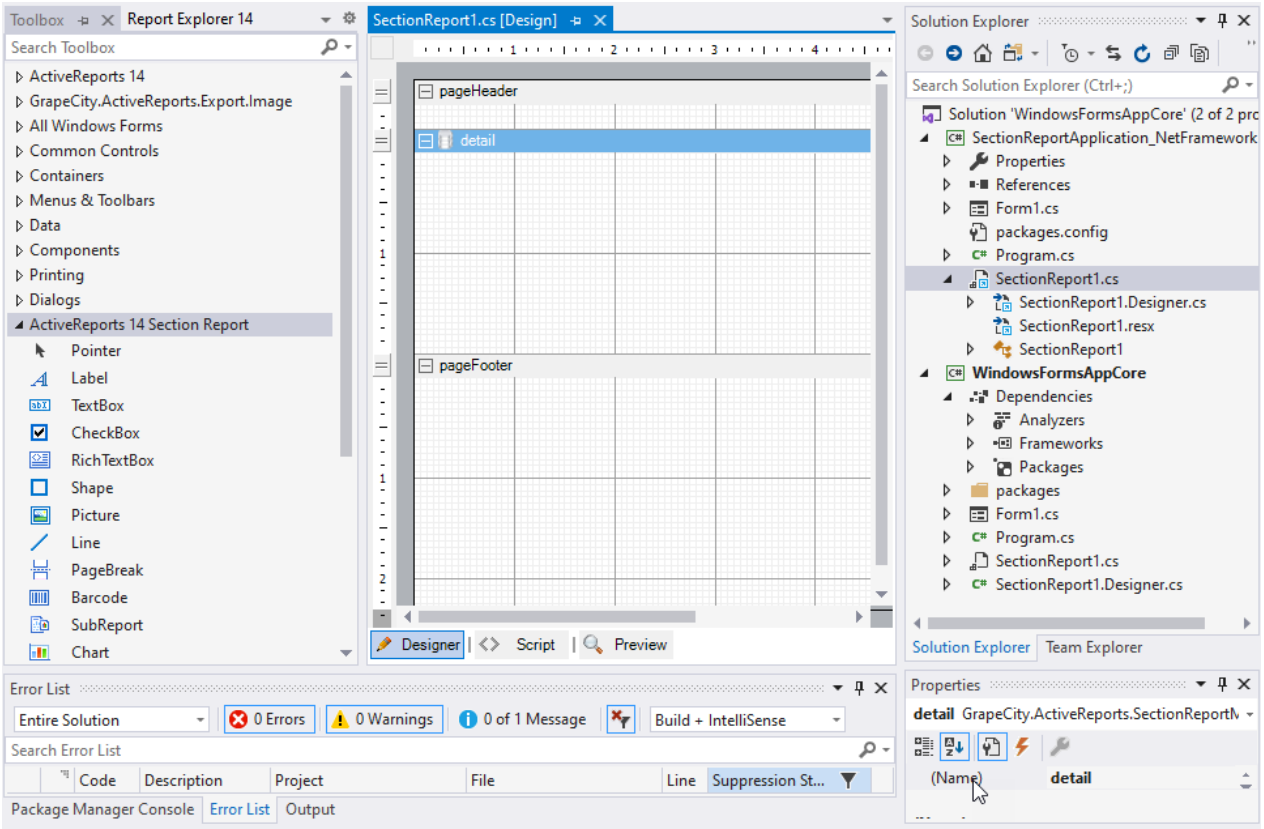
      and insert

      `<DependentUpon>SectionReport1.cs</DependentUpon>` as follows:

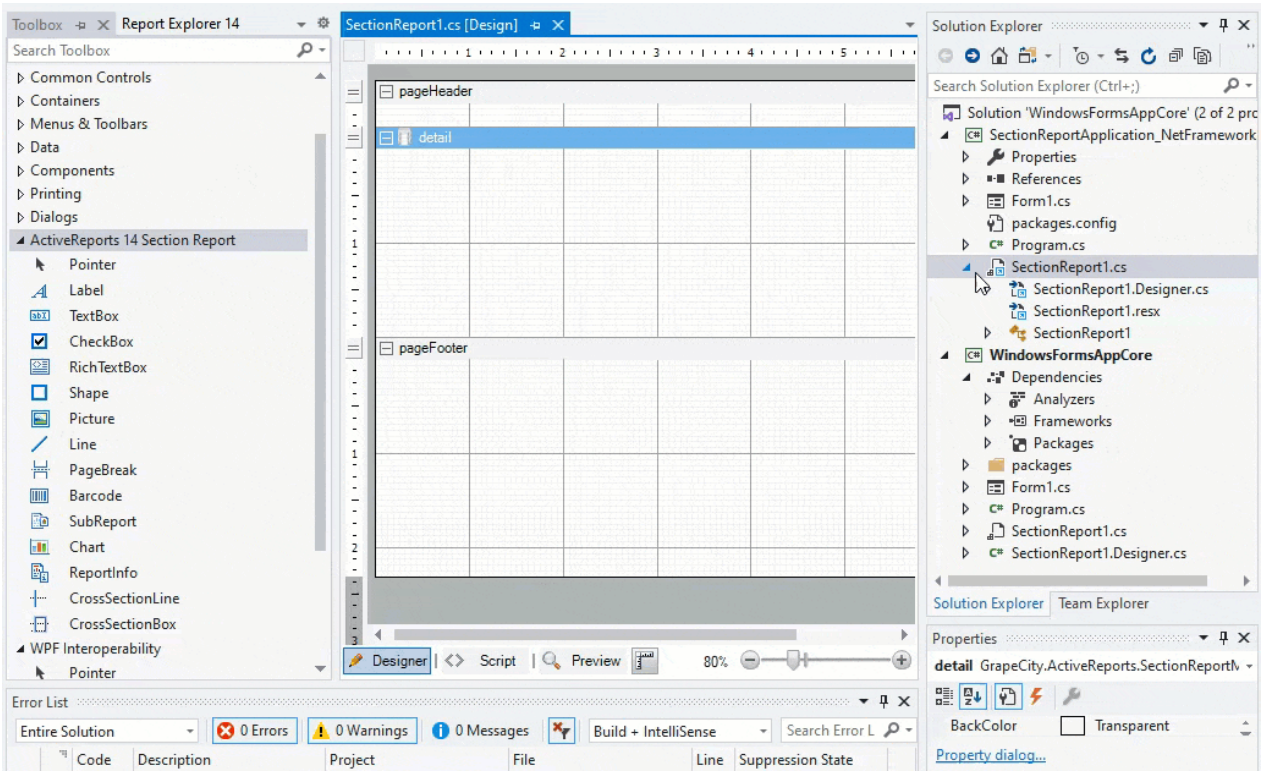
      ```
      <Compile Include="..\WindowsFormsAppCore\SectionReport1.Designer.cs">
            <Link>SectionReport1.Designer.cs</Link>
            <DependentUpon>SectionReport1.cs</DependentUpon>
      </Compile>
      ```

   > **Note**: If you do not find the .csproj file, you will need to first unload the project:
   >
   > 1. In the Solution Explorer, right-click the .NET Framework project node and select **Unload Project**.
   > 2. Again, right-click the project, select **Edit *projectname*.csproj**, and modify the project file as described before.

8. Double-click the linked SectionReport1.cs (in .Net Framework project) to open the ActiveReports Integrated Designer for the report.



9. Now design the report. The following gif shows how the modification of a report in .NET Framework project leads to modification of the report in the .NET Core project.



Scripting in Section Reports

In a section report, ActiveReports allows you to use VB.NET or C# script to port your custom logic to report layouts. This permits layouts saved to report XML (RPX) files to serve as stand-alone reports. By including scripting before you save the report layout as an RPX file, you can later load, run, and display the report directly to the viewer control without using the designer. In conjunction with report files, scripting allows you to update distributed reports without recompiling your project.

Script Editor

To access the script editor, click the script tab below the report design surface. The script tab contains two drop-downs (Object and Event).

- **Object**: Drop down the list and select one of the report sections, or the report itself.
- **Event**: Drop down the list and select from the list of events generated based on your selection in the Object drop down. If you select a report section as the Object, there are three events: Format, BeforePrint, and AfterPrint. If you select ActiveReport as the Object, there are seven events. See Section Report Events for further information.

Add script to the events in the same way that you add code to events in the code view of the report. When you select an event, the script editor generates a method stub for the event.

Using the **ScriptLanguage ('ScriptLanguage Property' in the on-line documentation)** property of the report, you can set the script language that you want to use.

Select the scripting language to use

- In design view of the report, click in the grey area below the report to select it.
- In the Properties window, drop down the ScriptLanguage property and select C# or VB.NET.

You can also add scripts at run time using the **Script ('Script Property' in the on-line documentation)** property.

> ⚠ **Caution**: Since the RPX file can be read with any text editor, use the **AddCode ('AddCode Method' in the on-line documentation)** or **AddNamedItem ('AddNamedItem Method' in the on-line documentation)** method to add secure information such as a connection string.

Tips for Using Script

- **Keep the section report class public**: If the Section Report class is private, the script cannot recognize the items in your report. The Section Report class is public by default.
- **Set the Modifiers property of any control referenced in script to Public**: If the control's **Modifiers** property is not set to **Public**, the control cannot be referenced in script and an error occurs when the report is run. The **Modifiers** property has a default value of **Private**, so you must set this property in the designer.
- **Use "this"** (as in C# code-behind) **or** "**Me"** (as in VB code-behind) **to reference the report.** Using "rpt" to reference the report is also possible but it is recommended to use the "this" and "Me" keywords.

  > **Note:** The basic approach of using the "this/Me" and "rpt" keywords is as follows - use "this/Me" to access the properties and controls added to the sections of the report, whereas use "rpt" within the instance of the ActiveReports class only to access its public properties, public events and public methods.

- **Use Intellisense support**: The script tab supports IntelliSense that helps in inserting the language elements and provides other helpful options when adding script to a report.

- **Use Run-time error handling**: When run-time errors occur, a corresponding error message is displayed in the Preview tab stating the problem.

> **Note:** A declared variable is not static by default. Use the **static** keyword to declare static variables.

Difference in script and code-behind event handler

Code-behind and the script tab require a different syntax for the event handler method definition. Use the **Private** modifier in code-behind and the **Public** modifier in the script editor.

See the following examples of the ReportStart event handler definition in Visual Basic and C#:

Script and code-behind examples in Visual Basic

The ReportStart event handler definition in the script editor:

```
Visual Basic.NET
Sub ActiveReport_ReportStart End Sub
```

The ReportStart event handler definition in code-behind:

```
Visual Basic.NET
Private Sub SectionReport1_ReportStart(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.ReportStart End Sub
```

Script and code-behind examples in C#

The ReportStart event handler definition in the script editor:

```
CS
public void ActiveReport_ReportStart() { }
```

The ReportStart event handler definition in code-behind:

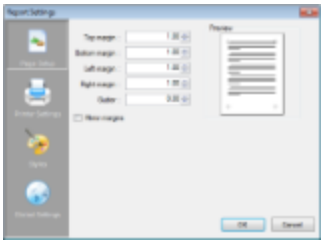
```
CS
private void SectionReport1_ReportStart(object sender, EventArgs e) { }
```

Report Settings Dialog

With ActiveReports, you can modify facets of your report, such as the page setup, printer settings, styles, and global settings at design time, as well as at run time. To make changes at design time, access the Report Settings dialog through any of the following:

- With the report selected, go to the Visual Studio toolbar select Report menu > **Settings**.
- In the Report Explorer, right-click the Settings node and select **Show** or double-click the Settings node.
- Click the gray area outside the design surface to select the report and in the Commands section at the bottom of the Properties Window, click the **Property dialog** command.

The Report Settings dialog provides the following pages where you can set or modify various settings of your report.



Page Setup

On the Page Setup page, you can make changes to the report margins (left, right, top, and bottom), specify a gutter, and select the Mirror margins option. This page also shows a preview of how each setting appears on the report page.

- **Top margin:** Set the Top margin for report pages.
- **Bottom margin:** Set the Bottom margin for report pages.
- **Left margin**: Set the Left margin for report pages.
- **Right margin**: Set the Right margin for report pages.
- **Gutter**: Set Gutter to give extra space between the edge of the page and the margins. This allows reports to be bound.
- **Mirror Margins**: Select this option to set same inner and outside margins for opposite pages in the report.

By setting a gutter and selecting Mirror margins, you can easily set up reports for publishing.

Printer Settings

On the Printer Settings page, you can make changes to the printer paper size and orientation.

- **Paper Size**: Select a paper size from the list of pre-defined paper sizes or choose Custom paper from the list to enable the Width and Height options for defining your own custom paper size.
- **Width**: Set the width of your custom paper size.
- **Height**: Set the height of your custom paper size.
- **Orientation**: Select one among Default, Portrait or Landscape as your paper orientation.
- **Collate**: Select whether to use collation or not.
- **Duplex**: Select whether the report should be printed in Simplex, Horizontal or Vertical duplex.
- **Paper Source**: Set the location of the paper source from the dropdown list.

> ⚠ **Important**: For items set to **Printer Default**, default printer settings are used from the printer installed on the environment where the report is being created. The paper size might also change based on the run time environment so in case the paper size of your report is fixed, please specify it beforehand.

Styles

On the Styles page, you can change the appearance of text associated with controls, either by creating a new style sheet, or by modifying and applying an existing style.

- **New**: Use this button to create a new style.
- **Delete**: Use this button to delete an existing style.
- **Export styles to file**: Use this button to export an existing style to an external XML *.reportstyle file.
- **Import styles from file**: Use this button to import styles a *.reportstyle file.
- **Font name**: Set or modify the font in your new or existing style.
- **Font size**: Set or modify the font size in your new or existing style.
- **Bold**: Enable or disable Bold text for your new or existing style.
- **Italic**: Enable or disable Italic text in your new or existing style.
- **Underline**: Enable or disable Underlined text in your new or existing style.
- **Strikethrough**: Enable or disable Strikethrough text in your new or existing style.
- **BackColor**: Set the Backcolor to use in your new or existing style.
- **ForeColor**: Set the Forecolor to use in your new or existing style.
- **Horizontal Alignment**: Set the horizontal alignment to Left, Center, Right, Justify for your new or existing style.
- **Vertical Alignment**: Set the vertical alignment to Top, Middle, Bottom for your new or existing style.
- **Script**: Select the script to use in your new or existing style.

Global Settings

On the Global Settings page, you can change the design layout of your report.

- **Snap Lines**: Select whether to use snap lines at design time or not.
- **Snap to Grid**: Select whether the control moves from one snap line to another at design time.
- **Show Grid**: Select whether to show or hide the grid at design time.
- **Grid Columns**: Set the count of columns in a grid.
- **Grid Rows**: Set the count of rows in a grid.
- **Dimension Lines**: Set whether to use dimension lines at design time or not.
- **Grid Mode**: Select whether to show gridlines as Dots or Lines.
- **Show Delete Prompt**: Select this option to get a warning when you try to delete a parameter or calculated field from the Report Explorer.
- **Ruler Units**: Set the ruler units in Inches or Centimeters.
- **Preview Pages**: Set the number of pages to display in the Preview tab. Minimum values is 1 and maximum is 10000 pages. By default, the Preview tab displays 10 pages.

  > **Note:** This property allows you to set number of pages to display in the Preview tab only. To set the number of pages for viewer/export, you can use **MaxPages ('MaxPages Property' in the on-line documentation)** property.

Date, Time, and Number Formatting

In Section Reports, you can set formatting strings for date, time, currency, and other numeric values using the OutputFormat property on the TextBox control. The OutputFormat dialog also allows you to select international currency values and select from various built-in string expressions. In addition to the built-in string expressions, you may use any .NET standard formatting strings. You can find information about these strings (Numerics and Date/Time formats) on MSDN.

> **Note**: The **ReportInfo** control has many preformatted options in the FormatString property for RunDateTime and Page Numbers. For more information, see Display Page Numbers and Report Dates.

> **Caution:**
>
> - The format from the OutputFormat property is applied to the value set in the DataField property of the Value property. It is not applied when a string is set in the Text property.
> - OutputFormat property settings are valid only for Double or DateTime type values. In case of a String or when no data type is set, the format is automatically applied to only those values that can be converted to Double or DateTime, otherwise no format is applied.

The OutputFormat property allows four sections delimited by a semicolon. Each section contains the format specifications for a different type of number:

- The first section provides the format for positive numbers.
- The second section provides the format for negative numbers.
- The third section provides the format for Zero values.
- The fourth section provides the format for Null or System.DBNull values.

For example: $#,#00.00; ($#,#00.00_); $0.00; #

Dates:

- dddd, MMMM d, yyyy = Saturday, December 25, 2012
- dd/MM/yyyy = 25/12/2012
- d or dd = day in number format
- ddd = day in short string format (for example, Sat for Saturday)
- dddd = day in string format (for example, Saturday)
- MM = month in number format
- MMM = month in short string format (for example, Dec for December)
- MMMM = month in string format (for example, December)
- y or yy = year in two digit format (for example, 12 for 2012)
- yyyy or yyyy = year in four digit format (for example, 2012)

Times:

- hh:mm tt = 09:00 AM
- HH:mm = 21:00 (twenty-four hour clock)
- HH = hours in 24 hour clock
- hh = hours in 12 hour clock
- mm = minutes
- ss = seconds
- tt = AM or PM

Currency and numbers:

- $0.00 = $6.25
- $#,#00.00 = $06.25
- 0 = digit or zero
- # = digit or nothing
- % = percent-multiplies the string expression by 100

> **Note:** Underscore (_) keycode can be used in **OutputFormat property** to skip the width of the next character. This code is commonly used as **_)** to leave space for a closing parenthesis in a positive number format when the negative number format includes parentheses. This allows both positive and negative values to line up at the decimal point.

Optimizing Section Reports

Optimization can be crucial for large reports (i.e. over 100 pages). Here is some information which will help you to achieve the best possible results for such reports. To optimize ActiveReports for the web, please refer to the memory considerations section.

Memory Considerations

- **Images:** Limit the use of large images when exporting to RTF and TIFF formats. Note that even one image uses a lot of memory if it's repeated on every page of a very long report exported to TIFF or RTF. If you are not exporting, or if you are exporting to Excel, PDF, or HTML, repeated images are stored only once to save memory, but the comparison necessary to detect duplicate images slows the processing time for the report.
- **SubReports**: Limit the use of subreports in repeating sections because each subreport instance consumes memory. For example, consider that a subreport in the Detail section of a report in which the Detail section is repeated 2,000 times will have 2,000 instances of the subreport. Nested subreports will compound the number of instances. If you need to use subreports in repeating sections, instantiate them in the ReportStart event instead of the Format event

of the repeating section so that they will be instantiated only once and use less memory.

- **CacheToDisk**: Set the CacheToDisk property of the Document object to True. Although it slows down the processing time, this allows the document to be cached to disk instead of loading the whole report in memory. The PDF export also detects this setting and exports the cached report. Please note that only the PDF export is affected by the CacheToDisk property; other exports may run out of memory with very large reports. By default, CacheToDisk uses IsolatedStorage, which requires IsolatedStorageFilePermission.
  It is recommended that you use the CacheToDiskLocation property to specify the physical path instead of using isolated storage so that you do not run into the size limit.
- **Summary**: Placing summaries (primarily page count and report totals) in header sections will have an adverse effect on memory as well as rendering speed with large reports using the CacheToDisk property. Since the rendering of the header is delayed until ActiveReports determines the total or page count of the following sections, CacheToDisk is unable to perform any optimization. The greater the number of affected sections, the longer rendering is delayed and the less optimization CacheToDisk will offer. Therefore, a group total in a group header section does not affect performance and memory as much as a report total in the report header.

Speed Considerations

- **Image:** An image repeated on every page of a very long report is stored only once to improve memory, but the comparison necessary to detect duplicate images slows performance. This is not only the case with the report document itself, but also with the Excel, PDF, and HTML exports as they perform their own comparisons.
- **Summaries:** Placing summaries (primarily page count and report totals) in header sections will slow report processing. ActiveReports must determine the total or page count of the following sections before it can render the header section. The greater the number of affected sections, the longer rendering is delayed. Therefore, a group total in a group header section does not affect performance and memory as much as a report total in the report header.
- **CacheToDisk:** Be sure that the CacheToDisk property of the Document object is not set to True. Setting it to True increases the amount of time the report takes to load, and should only be used with very large reports that use a lot of memory. If this is used with smaller reports of less than 100 pages, it may actually cause more memory to be used.
- **SELECT *** : Using the SELECT * statement is only recommended when you actually want to use all of the data returned by this statement. Contact your database administrator for other methods to speed up your queries.

Printing Considerations

- **Virtual Printer** : We recommend use of virtual printer in case report generate environment differs from the environment in which the report is viewed or printed like in the case of Web applications.
- **Line** : Please be careful as Line control has been used to draw borders within a report. An issue has been observed that spool size is increased (when comparing with Solid) during printing in case LineStyle property is set to an any value e.g. Dash or Dot etc. other than Solid that is the default value. As a result, time is taken for spooling by the report thus hindering the performance while printing. Same issue is observed when rendering a line using GDI+ in PrintDocument of .NET Framework.

CacheToDisk and Resource Storage

The CacheToDisk property of the SectionDocument object tells ActiveReports whether to hold resources in memory, or cache them somewhere on your disk. Caching resources slows processing time, but can save you from running out of memory with very large reports.

Isolated Storage

If you use the CacheToDisk property without setting a CacheToDiskLocation, the default location in which it caches resources is IsolatedStorage, so you must have IsolatedStorageFilePermission in order to use it. The cache capacity for IsolatedStorage may depend on your configuration, but does not exceed 3 GB.

> ⚠ **Important:** Temporary files and folders created in IsolatedStorage are not deleted automatically.

Cache to Disk Location

To avoid using IsolatedStorage, you can specify a folder in the CacheToDiskLocation property. The cache capacity for a disk location is 3 GB.

For an example of the code used to turn on CacheToDisk and specify a folder, see the **CacheToDiskLocation ('CacheToDiskLocation Property' in the on-line documentation)** property in the Class Library documentation.

Visual Query Designer

Visual Query Designer is a graphical interface that simplifies data binding by allowing users to interactively build queries and view the results. With the Visual Query Designer's interactive interface, users who are unfamiliar with SQL can easily design, edit and preview queries.

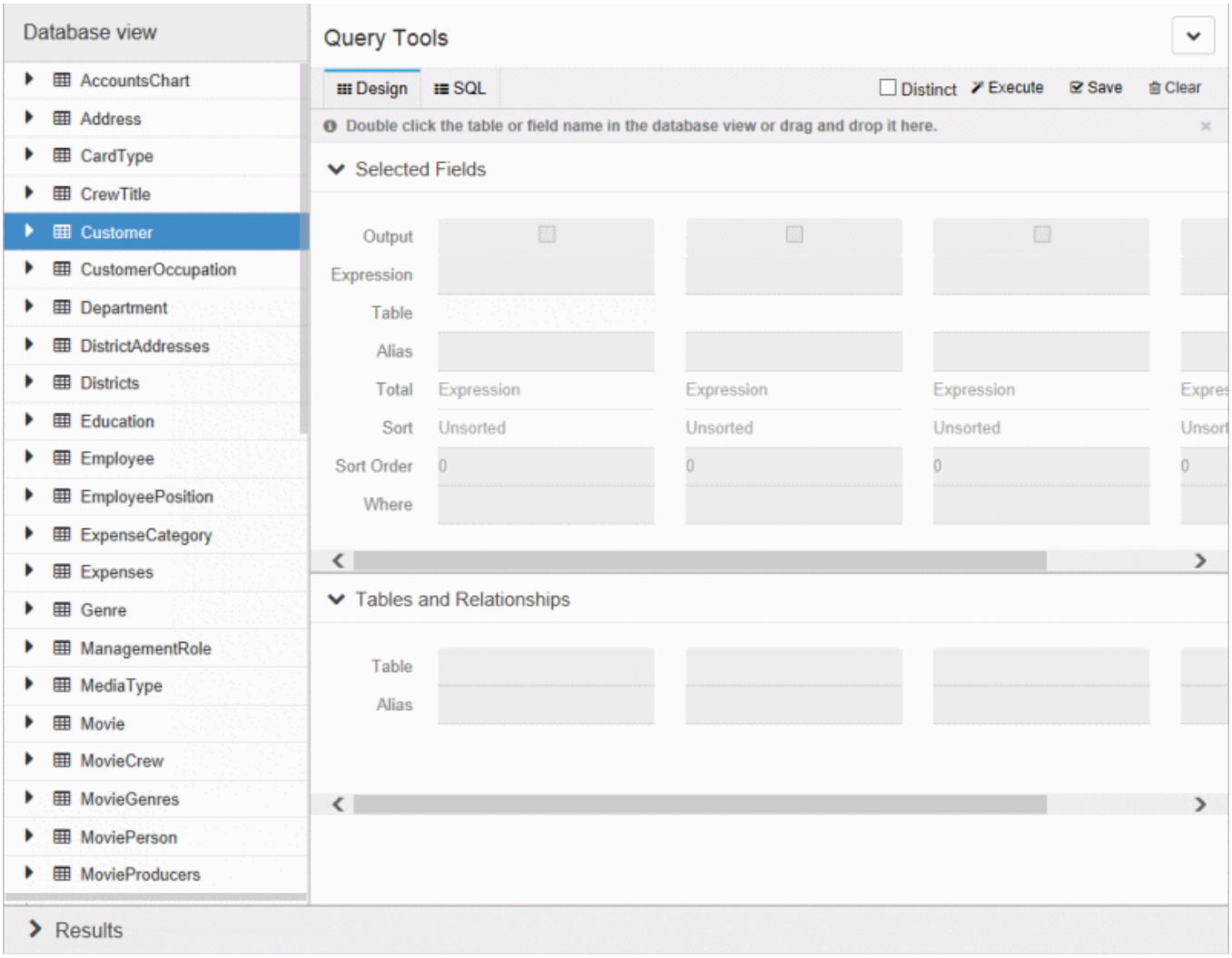
Visual Query Designer supports the following SQL capabilities:

- Selecting fields
- Custom expression
- Inner, left outer and right outer joins
- Filtering Data
- Grouping and aggregate functions
- Sorting
- Setting aliases for the selected fields and tables

For more information on how to use these capabilites in Visual Query Designer, refer to Query Building With Visual Query Designer.

> **Note**: You need to have Microsoft Internet Explorer 11 or higher installed on the system to run the Visual Query Designer.

See the graphic below to understand how a simple SQL query is generated in the Visual Query Designer.

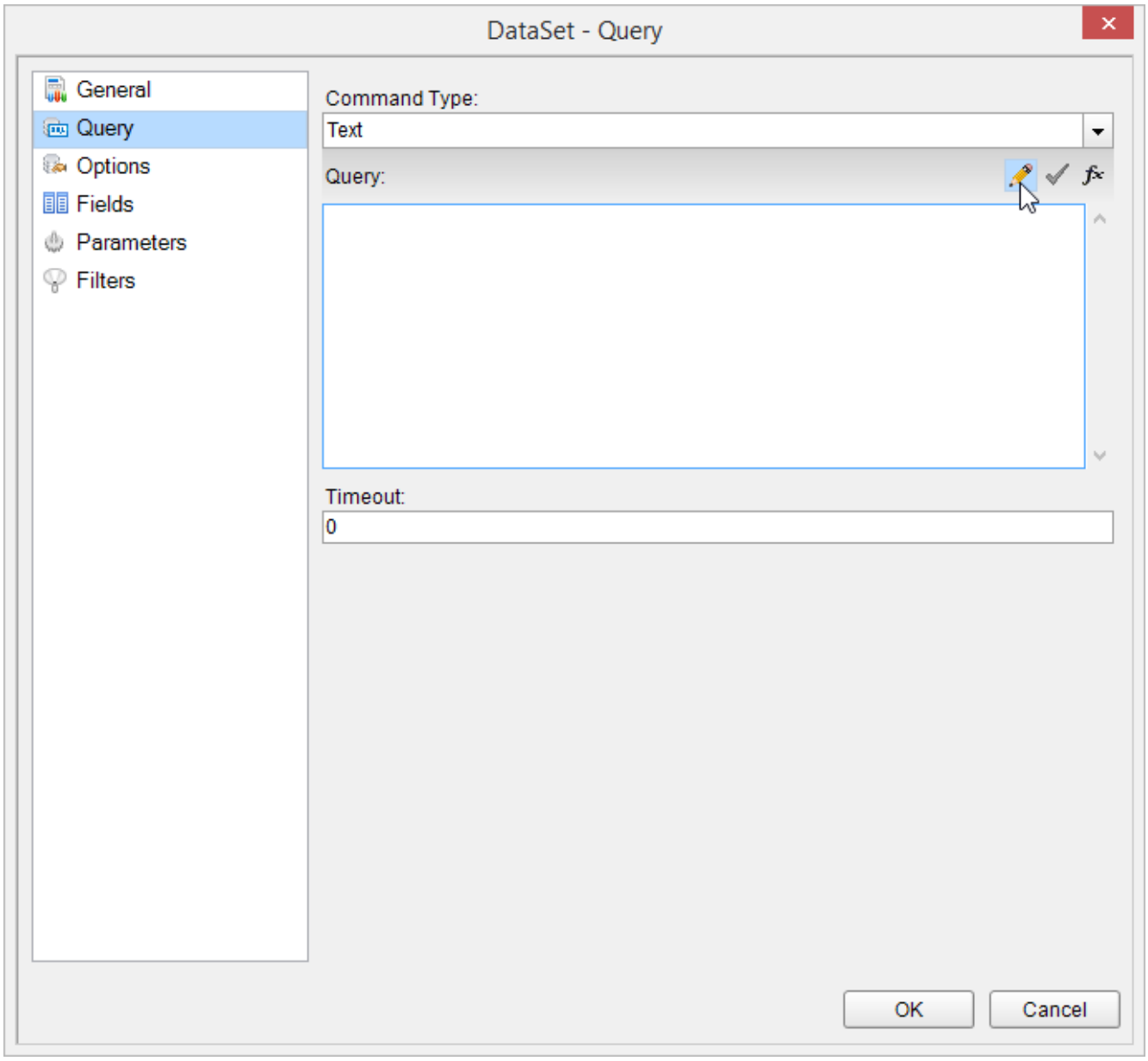


Limitations

2. Queries for XML, Object, DataSet Provider or any other specific data providers cannot be created in Visual Query Designer.
3. Unions, nested queries and stored procedures are not supported in **Design Tab**.
4. Crosses, full joins, provider-specific joins, and other SQL-specific implementation capabilities are not supported in the **Design Tab**.

Accessing the Visual Query Designer

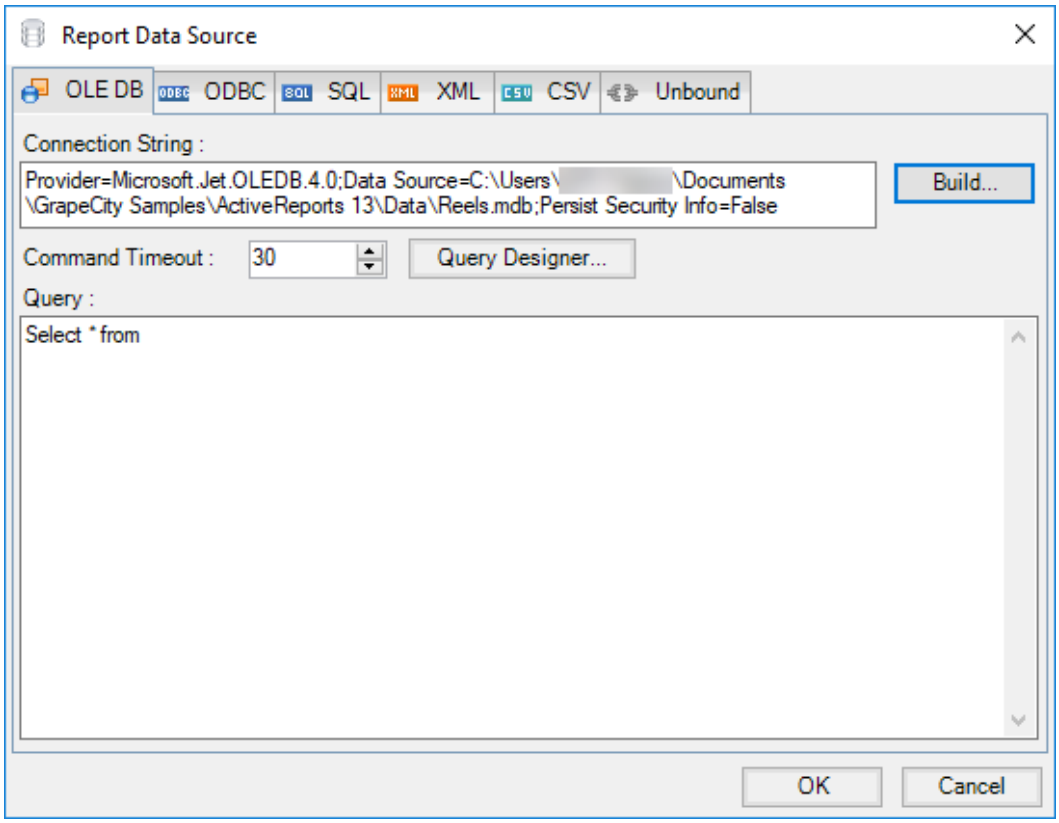
1. Connect a Page/RDL Report to a data source. See Connect to a Data Source for details on how to connect to a data source in Page/Rdl Reports and Bind Reports to a Data Source for Section Reports.
2. Right-click the data source node (DataSource1 by default) and select the Add Data Set option or select **Data Set** from the Add button on the Report Explorer toolbar to add a data set to the report.
3. In the DataSet Dialog that appears, select the Query page and then select the **edit with visual query designer button**  .

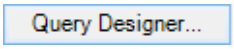


This opens the Visual Query Designer in a Page Report or Rdl Report.

In a Section Report

1. Connect a Section Report to a data source through the Report Data Source dialog. The **Query Designer** button is disabled until the report connects to a data source. For more information see, Bind Reports to a Data Source.



2. Once enabled, click the  button.

This opens the Visual Query Designer in a Section Report.

Elements of Visual Query Designer

Database View

Database View contains the structure of a database including namespaces, tables, views and columns. You can drag and drop

or double click the elements in the Database View to add them to the **Design** tab. Alternatively, you can double click the crossed arrows icon on the right hand side of each element in the Database View to add it to the Design tab.

This is the first step in query building through the Visual Query Designer. A SQL query is generated as you add the database elements to the **Design tab.**

Query Tools

The Visual Query Designer provides several tools to generate a query. The Query Tools section is divided into three major areas: Design tab, SQL tab and Toolbar buttons.

Design Tab

The Design tab is the area of the Visual Query Designer where you set up queries. It provides a visual interface for the SQL query you want to generate.

- **Selected Fields panel**

  Displays the fields, tables or any other element selected from the Database view. Each field in the Selected Fields panel has its own set of editable options.

Option	Description
Output	Checkbox to determine whether the field is included in the result set. The checkbox is selected by default when a field is added to the Selected Fields panel. You can clear this checkbox if you do not want the field to be displayed in the Results panel.
Table	Displays the name of the table the selected field belongs to.
Alias	Allows the user to provide an alternative name for the field.
Total	Applies grouping or aggregates on a field. Total (expression) is used to perform a calculation, retrieve the value of a control, define regulations, create calculated fields, and define a group level for a report. ◦ Expression - Allows selection of fields from a table. Custom expressions can also be specified here. ◦ GroupBy - Groups data based on the selected field. ◦ Count - Returns the number of items in a group. Implements the SQL COUNT aggregate. ◦ Avg - Returns the average of the values in a group. Implements the SQL AVG aggregate. ◦ Sum - Returns the sum of all the values in the group. Implements the SQL SUM aggregate. ◦ Min - Returns the minimum value in a group. Implements the SQL MIN aggregate. ◦ Max - Returns the maximum value in a group. Implements the SQL MAX aggregate. ◦ StDev - Returns the statistical standard deviation of all values in a group. Implements the SQL STDEV aggregate. ◦ Var - Returns the statistical variance of all values in the group. Implements the SQL VAR aggregate.
Sort	Arranges data in a prescribed sequence i.e. in Ascending or Descending order.
Sort Order	Allows the user to set the order of sorting in case multiple fields are to be sorted.
Where	Allows the user to set a filtering condition for the column data. The WHERE clause can be used when you want to fetch any specific data from a table omitting other unrelated data.

Note: When you add a table to the Selected Fields panel, all the fields in that table are added to the query. In effect

> you get a query like *Select * from Customers*.

- **Tables and Relationships**

  The Tables and Relationships panel displays a list of all the tables with fields in the **Selected Fields** panel. In case the Selected Fields panel has fields from multiple tables, a **Relations** button appears at the bottom of the related table's name to show the relationship between two tables.

  Tables and Relationships panel provides the following options for each table:

Option	Description
Table	Displays the names of all the tables with fields in the Selected Fields panel.
Alias	Allows the user to provide an alternative name for the table.

SQL Tab

The SQL Tab displays the SQL statement for the current query. Users can edit the query directly in the SQL Tab.

When you switch to the SQL Tab, the Visual Query Designer automatically formats your query in the correct syntax with highlighted keywords.

In the **SQL Tab** you can:

- Add new queries by directly typing SQL statements.
- Modify the SQL statement created by Visual Query Designer.

Tool Bar Buttons

Option	Description
Distinct Checkbox	Distinct Checkbox is used to remove duplicates from the result set of a SELECT statement. If checked, it allows users to display only distinct values.
Execute	Allows users to execute their query and display the result in Results panel.
Save	Allows users to save the query to a DataSet dialog.
Clear	Allows users to clear all the panels in the Visual Query Designer and the SQL tab along with it.

The Query Tools section also has a dropdown on the top right corner with two options:

1. **Toggle Panels**: To expand or collapse the **Selected Fields** and the **Tables and Relationships** panel.
2. **Show Hints**: To show or hide hints on how to use the Visual Query Designer effectively.
   Example: "Double click the table or field name in the database view or drag and drop it here." appears at the top of the **Selected Fields** panel.

Result panel

Displays the result of the query set in the Visual Query Designer.

This panel is populated when you click the **Execute** button on the Visual Query Designer toolbar after adding the required

fields or tables in the Selected Fields panel.

Query Building With Visual Query Designer

The topic takes you through the query building process in the Visual Query Designer. Query building in Visual Query Designer can be accomplished in a few simple steps:

Step 1: Adding fields from a table to generate a simple query

Step 2: Setting relationships (only applicable to queries using multiple tables)

Step 3: Setting options for individual fields or tables

Step 4: Executing a query

Step 5: Previewing a query

- **Create and Execute a Query (Single Table)**
- **Create and Execute a Query (Multiple Tables)**
- **Preview a Query**
- **Save a Query**
- **Edit a Query**
- **Clear a Query**
- **Access the Visual Query Designer**
- **Delete a field**
- **Sort data**
- **Display Distinct Values**
- **Aggregate Functions and Grouping**
- **Hide Fields from a Query**
- **Set filter condition**
- **Create a parameterized query**

The following steps assume that you have already added a Page Report or an Rdl Report template and connected it to a data source. See Quick Start and Connect to a Data Source for further information.
For more information on how to access Visual Query Designer, see Accessing Visual Query Designer.

> **Note**: This topic uses the Reels database. The Reels.mdb file can be downloaded from GitHub: *..\Samples14\Data\Reels.mdb*.

Create and Execute a Query (Single Table)

Go to Top

In the Visual Query Designer, you get a visual interface to assist you in quickly designing simple queries that reference a single table.

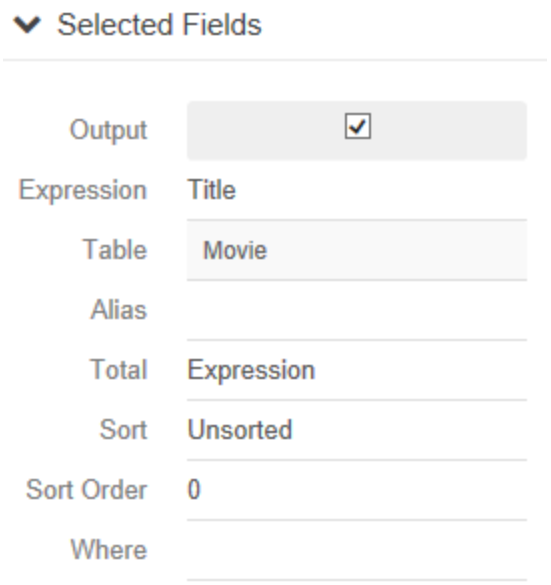
Query Result in SQL

SQL Query

```
select Movie.MovieID, Movie.Title
from Movie
```

Steps to create the Query in Visual Query Designer

1. From the Movie table in the **Database view**, drag and drop the field MovieID to the **Selected Fields** panel.



2. Add another field, Movie.Title from the same table, to the **Selected Fields** panel.
3. On the Toolbar of the Visual Query Designer, click the **Execute** button.

   Result data similar to the following appears in **Results** panel.

Create and Execute a Query (Multiple Tables)

Go to Top

In the Visual Query Designer, you get a visual interface to reference multiple tables and set up relationships between them.

The following example shows how to implement a right outer join using the tables Movie and MovieCrew from Reels database and apply a filter condition to your result set using the WHERE clause. For more information about table relationships and joins in Visual Query Designer, see Tables and Relations.

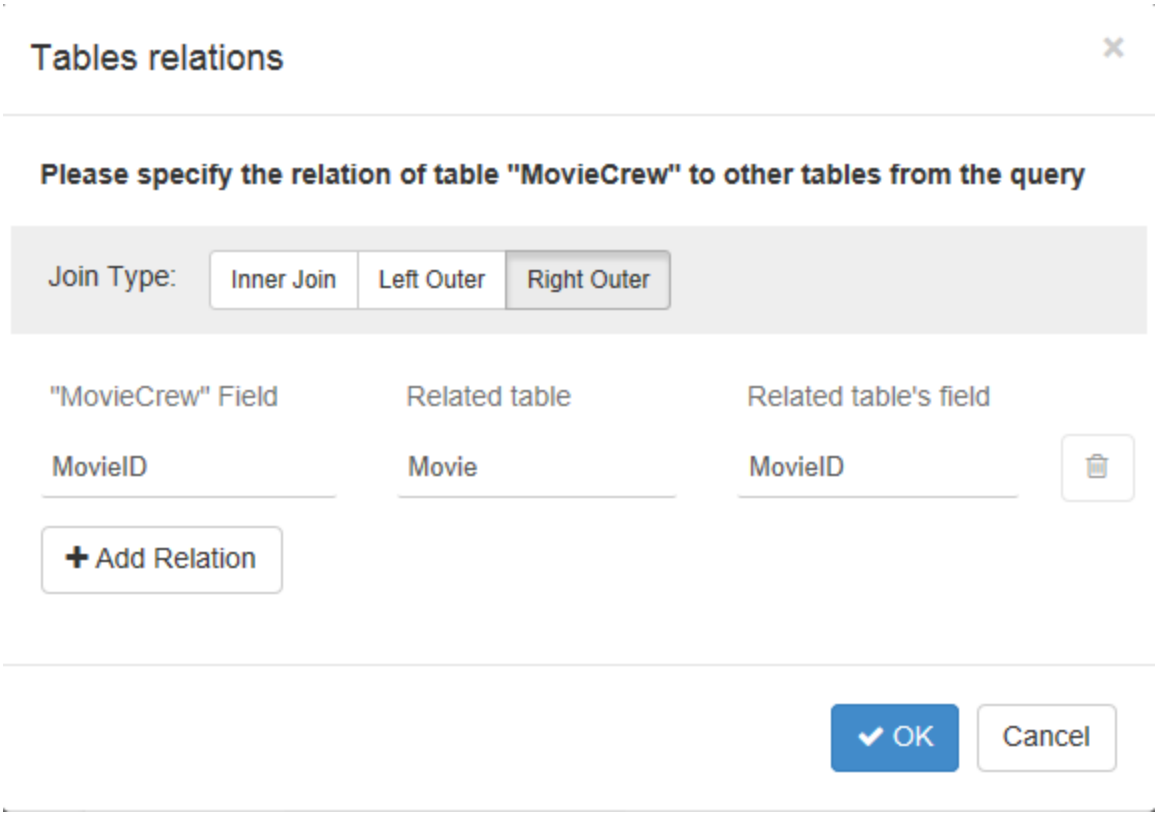
Query Result in SQL

SQL Query

```
select Movie.MovieID, Movie.Country, Movie.Title, MovieCrew.CastID,
MovieCrew.TitleID from Movie right join MovieCrew on
MovieCrew.MovieID = Movie.MovieID
where (Movie.Country = 'USA' and MovieCrew.CastID =1)
```

Steps to create the Query in Visual Query Designer

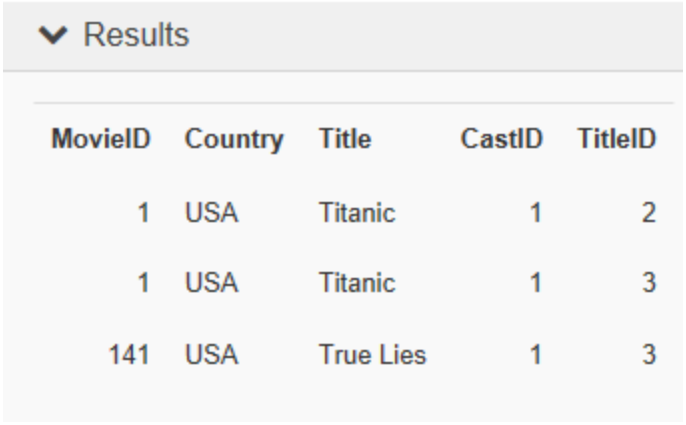
1. From the Movie table in the **Database view**, drag and drop the fields MovieID, Country and Title to the **Selected Fields** panel.
2. From the MovieCrew table in **Database view,** drag and drop the fields CastID and TitleID to the **Selected Fields** panel.
3. When you add the first field in step 2, a **Tables relations** dialog automatically appears on the screen.
   In **Tables relations** dialog, you can also select any other field from MovieCrew table which matches the related table's field to form a join between the both.
4. In **Tables relations** dialog, select the Right Outer Join Type for joining the two tables Movie and Movie Crew. The **Right Outer** tab is highlighted.
   Refer to Tables And Relations for more information on types of joins.



5. In **Tables relations** dialog, click **OK** to save the relationship between tables.
   Once the relationship has been set up between tables, you may also access the **Tables relations** dialog from the **Relations** button in the **Tables and Relationships** panel.
6. In the **Selected Fields** panel, under the Where option, add a filter condition for the Country field of the Movie

table. Set the value to "= 'USA'".
7. Again, in the **Selected Fields** panel, under the Where option, add a filter condition for the CastID field of the MovieCrew table. Set the value to "= 1".
8. On the Toolbar of the Visual Query Designer, click the **Execute** button.

   Result data similar to the following appears in **Results** panel.



Results

MovieID	Country	Title	CastID	TitleID
1	USA	Titanic	1	2
1	USA	Titanic	1	3
141	USA	True Lies	1	3

Preview a Query

Go to Top

When you have finished designing your query, you can execute it and then preview the result in the Visual Query Designer.

1. In the Query Tools section of the Visual Query Designer, go to the Toolbar.
2. Click the Execute button.

   You can preview the result in the **Results** panel at the bottom of the Visual Query Designer dialog.

Results

MovieID	Title
1	Titanic
2	Star Wars
3	Shrek 2

Note: When previewing a query, Visual Query Designer shows only a part of the data from the database.

Save a Query

Go to Top

Page Report/Rdl Report

Once a query is created in the Visual Query Designer, the **Save** button allows you to save the query into the **DataSet** dialog.

1. Once your query is created in Visual Query Designer, in the **Query Tools** section of the Visual Query Designer, go to the Toolbar.
2. Click the Save button. Your query appears in **Query** field of the **Query** page in the **DataSet** dialog.
3. Click **OK** to close the dialog.
   Your data set and queried fields appear as nodes in the Report Explorer.

Section Report

Once a query is created in the Visual Query Designer, the **Save** button allows you to save the query into the **Report Data Source** dialog.

1. Once the query is created in Visual Query Designer, in the **Query Tools** section of the Visual Query Designer, go to the Toolbar.
2. Click the Save button.
   Your query appears in **Query** field of the **Report Data Source** dialog.
3. Click **OK** to close the dialog. Your queried fields appear as bound field nodes in the Report Explorer.

Clear a Query

Go to Top

Once a query is created, the **Selected Fields** panel is populated with fields and **Tables and Relationship** panel displays the tables to which the fields used in the query belong.

1. In the **Query Tools** section of the Visual Query Designer, go to the Toolbar.
2. Click the Clear button.

   This clears the **Query Tools** section completely including **Selected Fields** and **Tables and Relationships** panel. It also removes the SQL query from the SQL tab and any data appearing in the **Results** panel.

Edit a Query

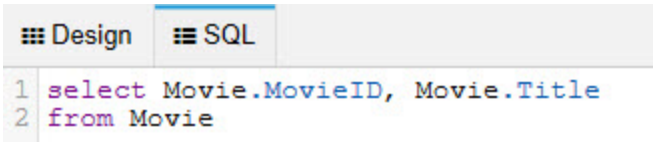
Go to Top

There are two ways to edit a query in the Visual Query Designer:

- Edit a query created in the Visual Query Designer directly in the SQL tab.

  Follow the steps below to edit a query created in the Visual Query Designer manually in the SQL:

1. From the Movie table in **Database view**, drag and drop the fields MovieID and Title onto the **Selected Fields** panel.
2. Under Query Tools, switch to the **SQL** tab to edit the query manually.



3. Enter the field name Movie.Length in the **SQL** tab.

```
SQL Query
select Movie.MovieID, Movie.Title, Movie.Length
from Movie
```

4. On the Toolbar of the Visual Query Designer, click the **Execute** button.

   The additional Length column appears in **Results** panel.

- Edit an existing SQL query in the Visual Query Designer.

  This approach has some limitations based on the type of queries that can be handled in the Visual Query Designer.

  Assuming that a query like the following already exists in the Data Set dialog of a Page/Rdl Report or the Report Data Source dialog of a Section Report, follow the steps below to edit it in the Visual Query Designer.

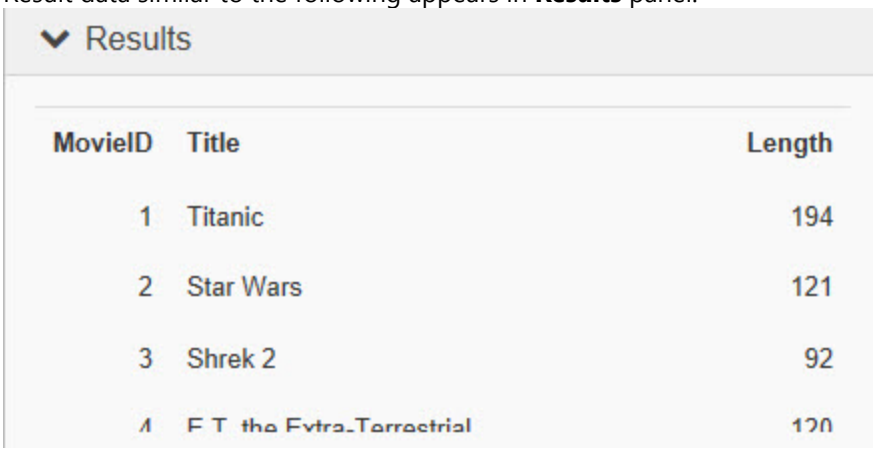
```
SQL Query
select Movie.MovieID, Movie.Title
from Movie
```

1. Open the Visual Query Designer and under **Query Tools**, go to the **SQL** tab. Notice that the SQL query is already present in the this tab.
2. Go to the **Design** tab and notice that the **Selected Fields** panel already contains the fields MovieID and Title.
3. From the **Database view**, drag and drop a field, Length from the Movie table onto the **Selected Fields** panel.
4. Go to the **SQL** tab again and see that the query now appears as follows:

```
SQL Query
select Movie.MovieID, Movie.Title, Movie.Length
from Movie
```

5. On the Toolbar of the Visual Query Designer, click the **Execute** button.

Result data similar to the following appears in **Results** panel.

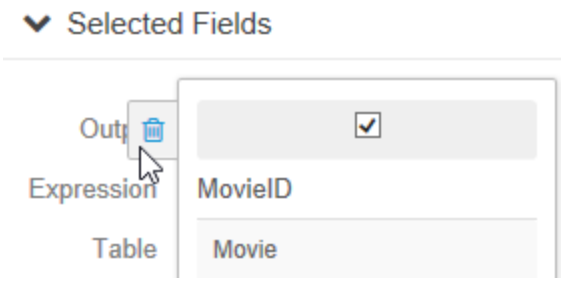


Delete a field

Go to Top

You can delete any field from a query in the Visual Query Designer. When you delete a field from a query, the field remains in the database, but is no longer used in the query.

1. From the Movie table in the **Database view**, drag and drop the fields MovieID, Country and Title onto the **Selected Fields** panel.
2. Hover your mouse over the MovieID field in **Selected Fields** panel to display the Delete icon.
3. Click the Delete icon to delete the field.
   Please note that once you delete a field, it is also removed from the SQL query in the **SQL** tab.



Sort data

Go to Top

You can sort the records in a table, query, form, or a report on one or more fields in the Visual Query Designer. For example, you can sort the Movie table by Title in ascending order and Country in descending order. In case multiple fields are being sorted, you can also determine which field is sorted first and which is sorted later.

Query Result in SQL

SQL Query

```
select Movie.MovieID, Movie.Title, Movie.Countryfrom Movieorder by Movie.Country desc,
Movie.Title asc
```

Steps to create the Query in Visual Query Designer

1. In the **Database view**, from the Movie table, drag and drop the fields MovieID, Title and Country onto the **Selected Fields** panel.
2. In the **Selected Fields** panel, go to the Title field and set the **Sort** option to ascending. **Sort Order** option is automatically set to 1.
3. Go to the Country field next and set the **Sort** option to ascending. **Sort Order** option is automatically set to 2. Based on steps 2 and 3, the table values sort on the Title field first and then on the Country field in ascending order.
4. In Country field, change the **Sort Order** value to 1. **Sort Order** value of the Title field automatically changes to 2. The table values now sort on the Country field first in and then on the Title field in ascending order.
5. In the Toolbar of the Visual Query Designer, click the **Execute** button.

   Result data similar to the following appears in **Results** panel.

Display Distinct Values

Go to Top

When retrieving data from a table, you may get duplicate records. Use the Distinct operator in the Select statement of your query to remove such values.

In the Visual Query Designer, you can use the Distinct checkbox available in the Toolbar to eliminate duplicate records. For example, you can retrieve unique records from YearReleased field of the Movie table.

Query Result in SQL

SQL Query

```
select DISTINCT Movie.Title, Movie.YearReleased
from Movie
```

Steps to create the Query in Visual Query Designer

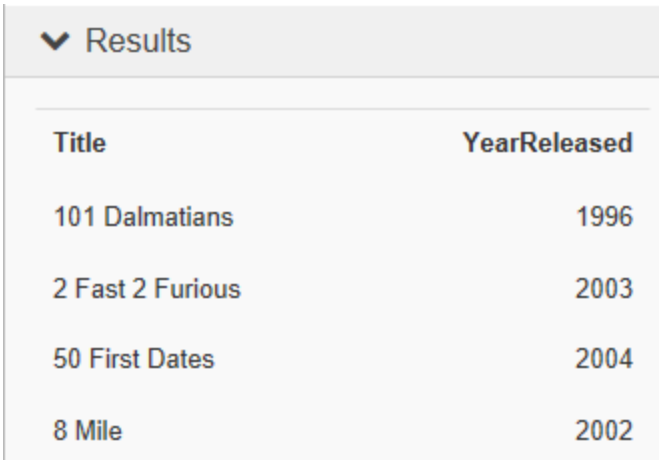
1. From the Movie table in the **Database view**, drag and drop the Title and YearReleased fields onto the **Selected**

Fields panel.

2. In the Toolbar of the Visual Query Designer, check the 

checkbox to display unique values from the YearReleased field.
3. Click the **Execute** button.

   Result data similar to the following appears in **Results** panel.



Aggregate Functions and Grouping

Go to Top

You can group data on a field and create an aggregate query that involves a function such as Sum or Avg in the Visual Query Designer. For example, you can group the movies in the Movie table by Country and calculate the average movie ratings for different countries using the Visual Query Designer.

Query Result in SQL

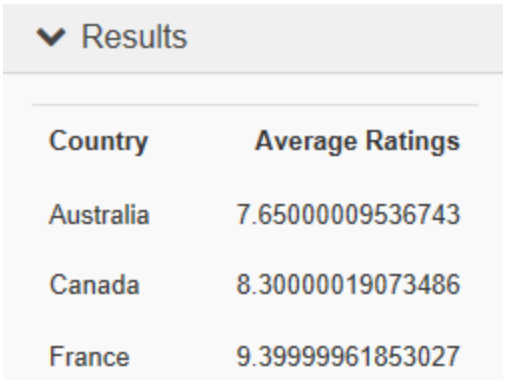
SQL Query

```
select Movie.Country, Avg(Movie.UserRating) as [Average Ratings]
from Movie group by Movie.Country
```

Steps to create the Query in Visual Query Designer

1. From the Movie table in the **Database view**, drag and drop the Country and UserRating fields onto **Selected Fields** panel.
2. In the **Selected Fields** panel, under the Country field, select **GroupBy** from the **Total** dropdown list. This groups the data by country name.
3. In the **Selected Fields** panel under the UserRating field, select the option Alias and set its alternate name to **Average Ratings**.
4. Under the UserRating field again, select **Avg** from a list of pre-defined aggregate functions in the **Total** dropdown list. This provides average user ratings for movies.
5. In the Toolbar of the Visual Query Designer, click the **Execute** button.

   Result data similar to the following appears in **Results** panel**.**



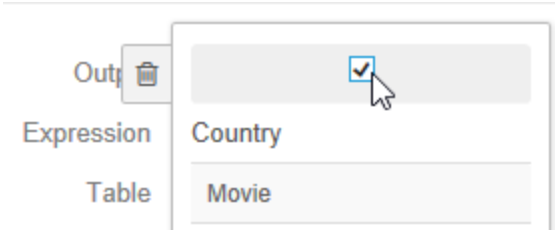
Hide Fields from a Query

Go to Top

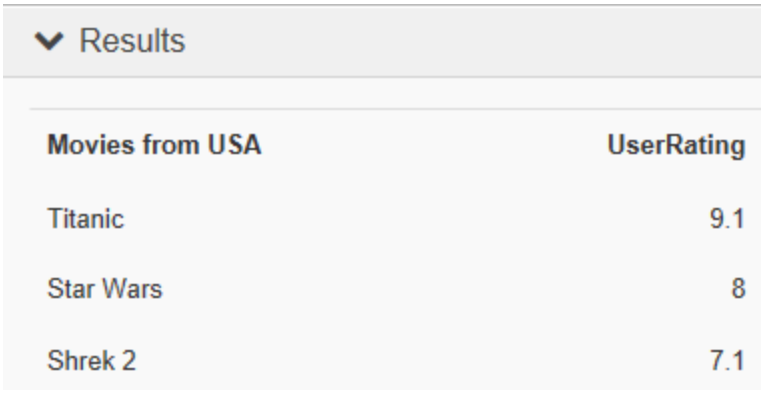
The Hide option in the Visual Query Designer allows you to hide part of the data that a query retrieves. For example, you can hide the MovieID field in the Movie table, from the result set of your query.

Follow the steps below to hide a field through the Visual Query Designer:

1. From the Movie table in the **Database view**, drag and drop the fields Title, UserRating and Country onto the **Selected Fields** panel.
2. In the **Selected Fields** panel, go to the Country field and select the **Where** option and set it's value to " = 'USA' "
3. In the **Selected Fields** panel, go to the Title field and set the value of the Alias option to **Movies from USA**.
4. Clear the **Output** check box for the Country field as shown in the image below to hide the field from the result set.



5. In the Toolbar of the Visual Query Designer, click the **Execute** button.



The data for the Country field does not appear in the in the **Results** panel anymore.

Set filter condition

Go to Top

The SQL **Where** clause is used to filter results that match a given criteria. The Where clause can be used when you want to fetch any specific data from a table omitting other unrelated data.

For example if you want to display UserRating of only those movies where the MovieID is either 1 or 2, you can use the **Where** clause with an '=' operator in the Visual Query Designer.

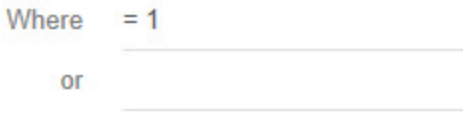
Query Result in SQL

SQL Query

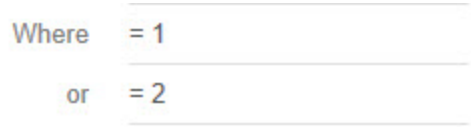
```
select Movie.MovieID, Movie.UserRating
from Movie
where (Movie.MovieID = 1 or Movie.MovieID = 2)
```

Steps to create the Query in Visual Query Designer

1. Form the Movie table in **Database view,** drag and drop the MovieID and UserRating fields onto the **Selected Fields** panel.
2. In the **Selected Fields** panel under the MovieID field, select the **Where** option and set the value to "=1".

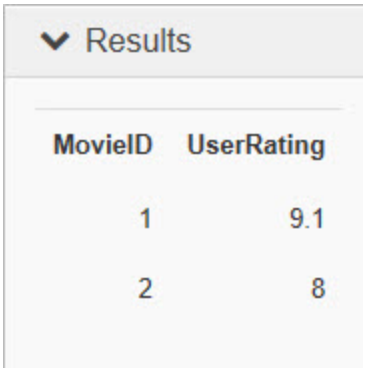
   

3. Add an **OR** condition in MovieID field and set the value to "=2".

   Rows which have either have MovieID = 1 or MovieID = 2 are displayed in the Result set.
4. In the Toolbar of the Visual Query Designer, click the **Execute** button.

   Result data similar to the following appears in **Results** panel**.**

Create a parameterized query

Go to Top

You can set parameters in your query using the Visual Query Designer. A parameterized query generally prompts the user to enter a value before the query is executed, to determine the type of data to be displayed in the result set.

As an example of a simple parameterized query, you can create a query parameter that prompts a user for a Movie ID and displays the Title, UserRating and Length of the movie based on the ID entered.

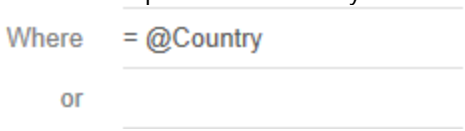
Query Result in SQL

SQL Query

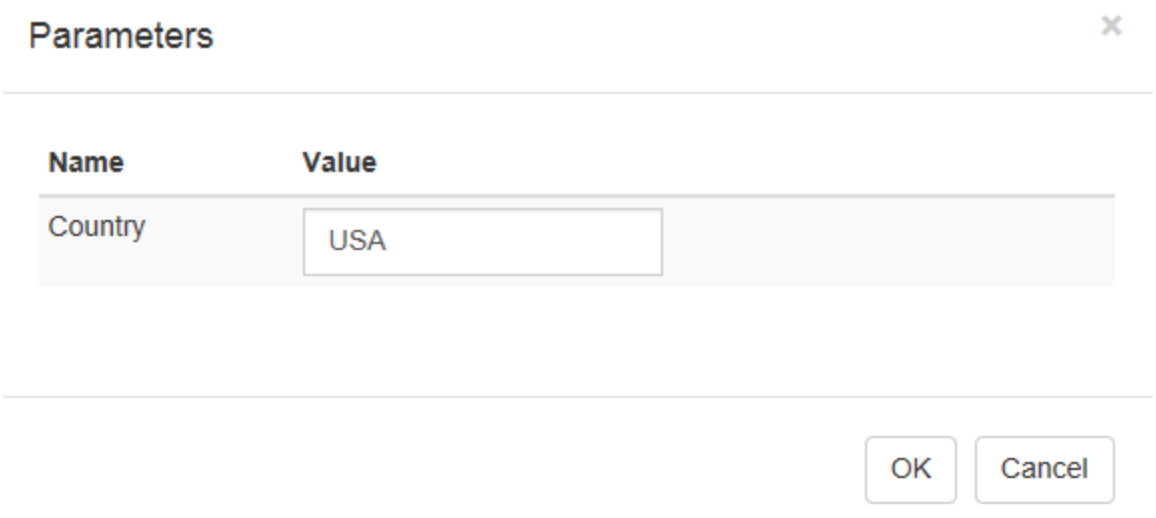
```
select Movie.MovieID, Movie.Title, Movie.Country, Movie.UserRating
from Movie
where Movie.Country = ?
```

Steps to create the Query in Visual Query Designer

1. From the Movie table in the **Database view**, drag and drop the MovieID, Title, UserRating and Country fields onto the **Selected Fields** panel.
2. In the **Selected Fields** panel under the Country field, select the option **Where** and set its value to "= @Country". This creates a parameter Country.



3. In the Toolbar of the Visual Query Designer, click the **Execute** button. The **Parameters** dialog automatically appears on the screen.



4. Enter the parameter value USA in the dialog box and click **OK.**

   Result data similar to the following appears in **Results** panel.

Results

MovieID	Title	Country	UserRating
1	Titanic	USA	9.1
2	Star Wars	USA	8
3	Shrek 2	USA	7.1

Go to Top

Tables And Relations

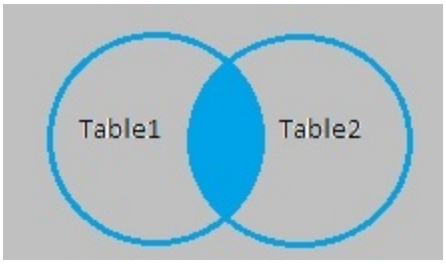
Queries can incorporate fields from different tables. It is the relationship you set up between the data in these tables that determines how the data appears in the result set.

Users can set up these relationships between tables using SQL Joins like Inner Join, Left Join and Right Join in the Visual Query Designer.

1. **Inner Join (simple join)** - Inner Join matches rows from Table1 with rows in Table2, and allows the user to retrieve records that show a relationship in both the tables. Inner join produces a set of data that matches both Table1 and Table2.

   Syntax for the SQL Inner Join:
   SELECT columns
   FROM table1
   INNER JOIN table2
   ON table1.column = table2.column;

   Visual Illustration

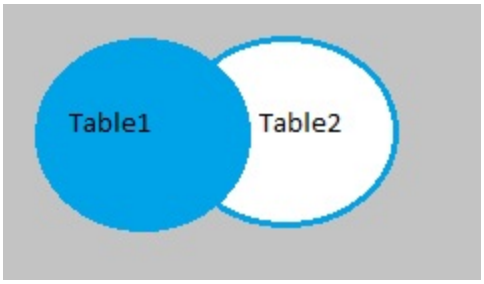
   

   SQL Inner Join returns the records where table1 and table2 intersect.
2. **Left Outer Join (left join)** - Left Outer Join allows users to select rows that match from both the left and right tables, plus all the rows from the left table (table 1). This means only those rows from table2 that intersect with table1 appear in the result set.

   Syntax for SQL Left Outer Join:
   SELECT columns
   FROM table1
   LEFT [OUTER] JOIN table2

ON table1.column = table2.column;

Visual Illustration

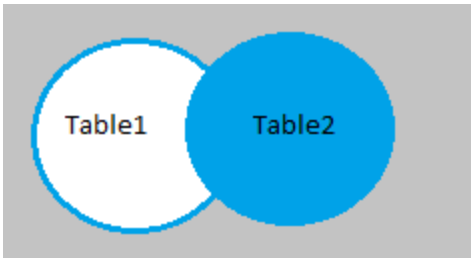


SQL Left Outer Join returns the records from table1 and only those records from table2 that intersect with table1.

3. **Right Outer Join (right join) -** Right outer join allows users to select rows that match from the both the left and right tables, plus all the rows from right table (table 2). This means that only those rows from table 1 that intersect with table 2. appear in the result set

Syntax for SQL Right Outer Join:
SELECT columns
FROM table1
RIGHT [OUTER] JOIN table2
ON table1.column = table2.column;

Visual Illustration



SQL Right Outer Join returns the records from table 2 and only those records from table 1 that intersect with table2.

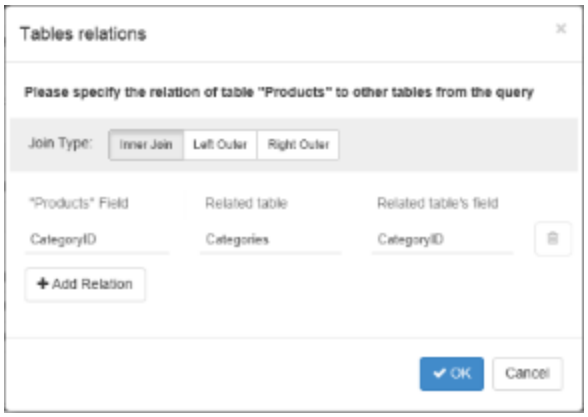
Tables relations dialog

The **Tables relations** dialog allows users to set up a relationship between two different tables with at least one common field.

Complete the following steps to access the **Tables relations** dialog:

1. In the **Visual Query Designer**, drag and drop a field or fields from a table in the Database view to the Selection Fields panel.
2. Add another field from a different table in the Database tab to the Selected Fields panel. Make sure that at least one field between these two tables' matches i.e. the second table contains a foreign key.
3. When the field in step 2 is added, the **Tables relations** dialog automatically pops up on the screen.

Once the relationship has been set up between tables, you may also access the **Tables relations** dialog from the **Relations** button in the **Tables and Relationships** panel.



Option	Description
Join Type	Enables selection of an appropriate Join type out of Inner Join, Left Outer Join and Right Outer Join. Example: Inner Join tab is highlighted in the image above.
<Table Name> Field	Displays the name of the field that is common between tables i.e. the foreign key name in the second table. Example: "Products" Field contains the 'Category ID' field in the image above.
Related Table	Displays the name of the table to which the relationship has been set up. Example: 'Categories' table is listed in the image above.
Related Table's Field	Displays the name of the field from the table to which the relationship has been set up. Example: 'Category ID' is the field from the Categories tables listed in the image above.
Delete	Icon adjacent to the Related Table's Field to delete the currently added relation.
Add Relation	Button that allows users to add another relationship to the table.
Cancel	Closes the Tables relations window.
OK	Saves the relationship between tables as a SQL query in the SQL tab.

Go to Top

Using the Visual Query Designer

The following walkthrough shows how to implement an Inner Join using the tables Categories and Products from the Nwind database. It also explains how to use a parameter and sort data in a Visual Query Designer. You may also see Tables and Relations for more information on table relationships and joins in Visual Query Designer.

Note:

- This walkthrough uses the **Products and Categories** table from the NWIND database. By default, in ActiveReports, the NWIND.mdb file is located in [User Documents folder]\GrapeCity Samples\ActiveReports 14\Data\NWIND.mdb.
- This walkthrough uses Page Reports, but the same steps can be used to generate queries in Rdl or Section reports also. See Accessing the Visual Query Designer, for more information on how to access the Visual Query Designer in Rdl and Section reports.

- **Add an ActiveReport to a Visual Studio Project**
- **Access the Visual Query Designer**
- **Create a Query**
- **Save a Query**
- **View the Report**

Add an ActiveReport to a Visual Studio project

1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.
3. In the **Add New Item** dialog that appears, select ActiveReports 14 Page Report and in the **Name** field, rename the file as VisualQueryDesigner.
4. Click the **Add** button to open a new Page Report in the designer.

See Quick Start for information on adding different report layouts.

Access the Visual Query Designer

1. Connect a Page Report to a data source. See Connect to a Data Source for details on how to connect to a data source in Page Reports.
2. Right-click the data source node (DataSource1 by default) and select the Add Data Set option or select **Data Set** from the Add button on the Report Explorer toolbar to add a data set to the report.
3. In the DataSet Dialog that appears, select the Query page and then select the **Edit with visual query designer** button  .
   This opens the Visual Query Designer.

Create a Query

Visual Query Designer provides you with an interface to reference multiple tables, set up relationships, sort the data and add parameters to your query.
Follow the steps below to create a query like the following using the Visual Query Designer.

Query Result in SQL

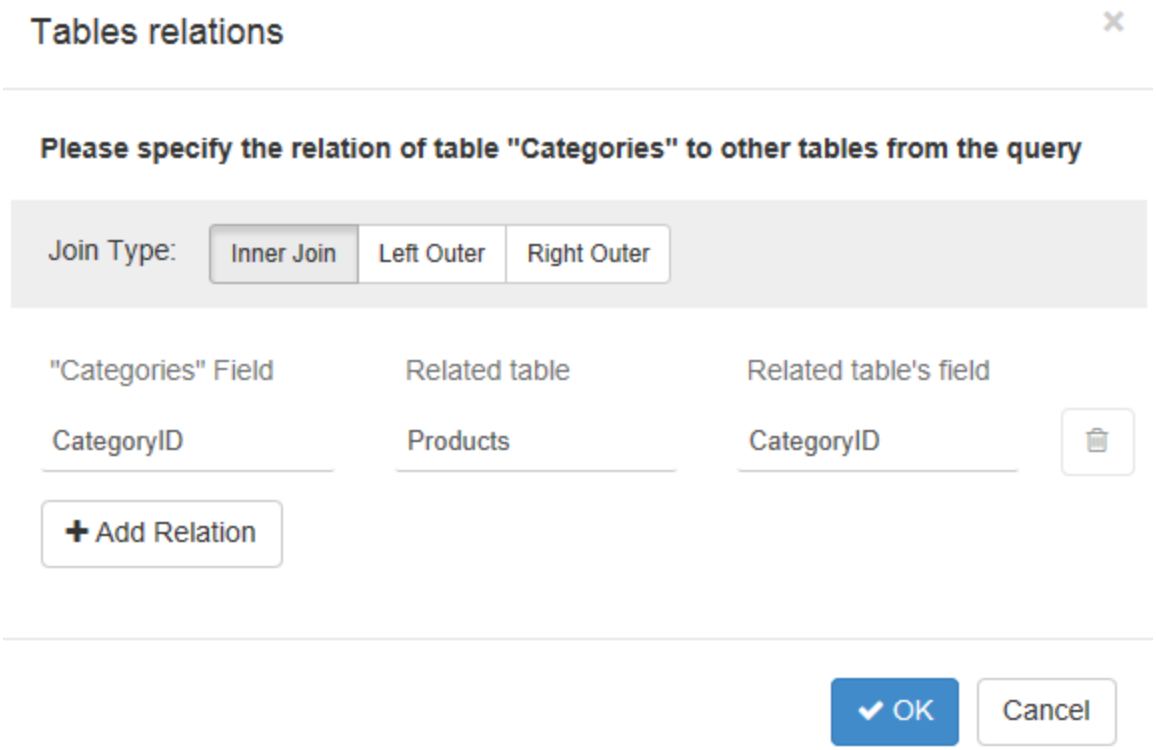
SQL Query

```
select Products.ProductName, Products.UnitPrice, Categories.CategoryName
from Products inner join Categories on Products.CategoryID = Categories.CategoryID
where Categories.CategoryName = ?
order by Products.UnitPrice desc
```

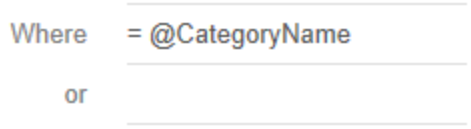
Steps to create a Query in Visual Query Designer

1. From the Products table in the **Database view**, drag and drop the fields ProductName and UnitPrice to the **Selected Fields** panel.
2. From the Categories table in **Database view,** drag and drop the field CategoryName to the **Selected Fields** panel.
3. When you add the field in step 2, a **Tables relations** dialog automatically appears on the screen.
   In the **Tables relations** dialog, you can also select any other field from Categories table which matches the related table's field to form a join between the both.

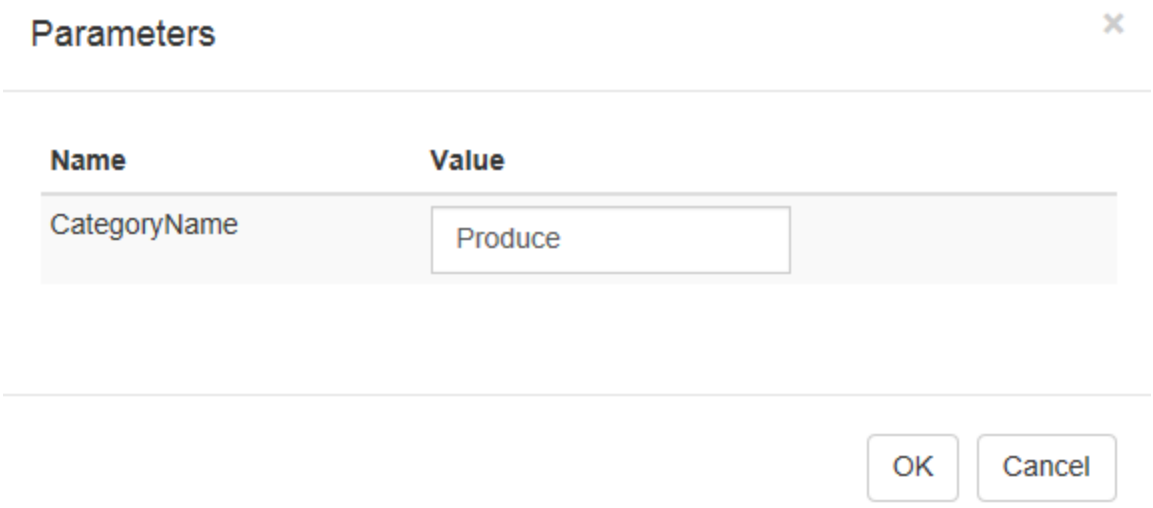
4. In **Tables relations** dialog, select the Inner Join Type for joining the two tables Products and Categories. The **Inner Join** tab is selected by default.
   Refer to Tables And Relations for more information on types of joins.



5. In **Tables relations** dialog, click **OK** to save the relationship between tables.
   Once the relationship has been set up between tables, you may also access the **Tables relations** dialog from the **Relations** button in the **Tables and Relationships** panel under the **Query Tools** section.
6. In the **Selected Fields** panel under the CategoryName field, select the option **Where** and set its value to "= @CategoryName". This creates a parameter on the CategoryName field.

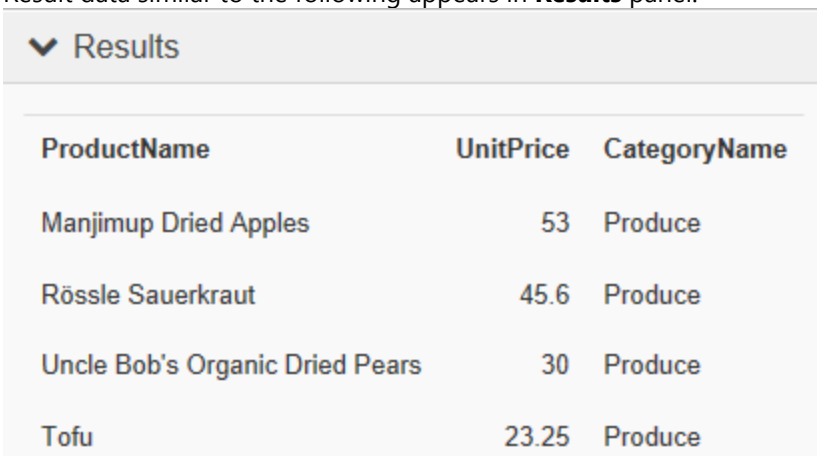


7. In the **Selected Fields** panel, go to the UnitPrice field and set the **Sort** option to descending. This sorts the data in descending order on UnitPrice.
8. On the Toolbar of the Visual Query Designer, click the **Execute** button. A **Parameters** dialog appears on the screen.



9. Enter any parameter value, for example, Produce in the dialog box and click **OK.**

   Result data similar to the following appears in **Results** panel.



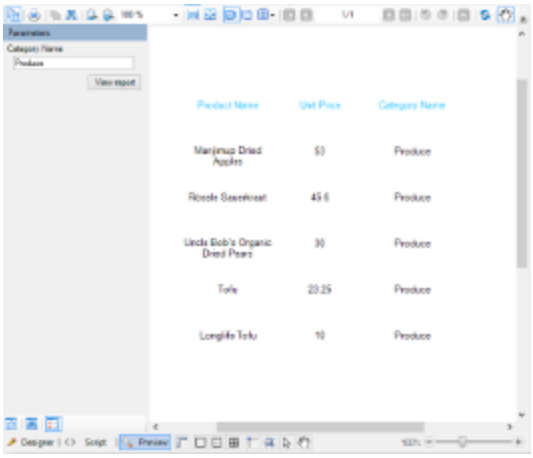
Save a Query

1. Once your query is created in Visual Query Designer, go to the Toolbar in the **Query Tools** section of the Visual Query Designer.
2. Click the Save button. Your query appears in the **Query** field of the **Query** page in the **DataSet** dialog.

   Note: On clicking the **Save** button, your query is automatically validated by the Visual Query Designer.

View the Report

1. Place a data region like a Table onto the design surface and add fields to it. For more information on how to add fields to a table, see the **Adding Data** on the Table data region page.
2. Click the preview tab to view the report with the Parameters panel displayed in the sidebar.



OR

Open the report in the Viewer to view the report with the Parameters panel displayed in the sidebar. See Windows Forms Viewer for further information.

Interactive Features

ActiveReports supports features like parameters, filters, drill-down, links, document map and sorting to provide an interactive look to your report at run time.

Parameters

ActiveReports allows you to set parameters in your report to filter the data displayed. Parameters make navigation of the report easier for the user at run time. See Parameters for further details.

Filters

The filtering feature is only available with page layout reports. By using filters in your page report or RDL report you can limit the information you want to display on your report. See Filtering for further details.

Drill-Down Reports

When you open a report with drill-down features, part of the data is hidden so that you only see high-level data until you request more detail. See Drill-Down Reports for more information.

Bookmark, Hyperlinks and Drill-Through Links

Bookmark Links

When you click a bookmark link, the viewer navigates to a bookmarked item within the current report.

Hyperlinks

When you click a hyperlink, your machine's default internet browser opens to display a Web page.

Drill-Through

Using the drill-through link feature in your report you can navigate to another report for details about the item you clicked.

See Linking in Reports for further details.

Document Map

The Document Map (Table of Contents) feature allows you to navigate to a particular item in a report. See Document Map for further details.

Sorting

The sorting feature allows you to organize your data and present it in a logical order at run-time. Using this feature you can sort the data alphabetically or numerically in ascending or descending order. See Sorting for further details.

> **Note:** You cannot use the interactive features in the following cases:
>
> - With Adobe Acrobat Reader and RawHTML types of the WebViewer.
> - With reports that use the collation feature, i.e. reports that have two or more themes.

Annotations

The annotations feature allows you to add floating text bars or images to call attention to specific items or values or to add notes and special instructions directly to the reports. These annotations are accessible through the Annotation button present on the Viewer toolbar which is hidden by default. Annotations added via the viewer's toolbar are temporary and are destroyed when the report closes. See Annotations for further details.

Parameters

ActiveReports allows you to use parameters to filter or add the data to display in reports at run time. You can either prompt users for parameters so that they control the output, or supply the parameters behind the scenes.

Adding parameter for different data sources

A query parameter can get its value from the Report Parameters collection (entered by the user or from a value you supply), a field in another dataset, or an expression. Syntax for adding a parameter in your query might differ depending upon the data source that you are using. Use the syntax specific to your data source type to create a parameter.

Parameterized query for different data sources are as follows:

Data Source	Parameter Syntax	Example
OleDB	(?)	SELECT * FROM Customer WHERE (CustomerID = ? AND AccountNumber = ?)
ODBC	@ParameterName	SELECT * FROM Customer WHERE (CustomerID = @CustomerID AND AccountNumber = @AccountNumber)
SQL Client	@ParameterName	SELECT * FROM Customer WHERE (CustomerID = @CustomerID AND AccountNumber = @AccountNumber)
OracleDB	:ParameterName	SELECT * FROM Customer WHERE CustomerID = :CustomerID AND AccountNumber = :AccountNumber

Page Report/RDL Report

In a page report or a RDL report, the easiest way to build queries with parameters is to use the Visual Query Designer, as it automatically sets up each parameter.
In the DataSet dialog, click on  to access Visual Query Designer for creating SQL queries. See Query Building With Visual Query Designer for further information on how to create a parameterized query using the interactive query designer.

However, if you would like to do it manually, you must enter each parameter in three locations: the Report Parameters dialog (for filtering data at run time), the Parameters page of the DataSet dialog, and the Query page of the DataSet dialog.

Report - Parameters dialog

The Report - Parameters dialog allows you to control how and whether a user interface is presented to your users for each parameter. You have to set the following properties in the dialog to create a parameter:

- Enter a report parameter name. Each report parameter in the collection must have a unique name, and the name must match the name you call in the Parameters page of the DataSet dialog. In the example above, the name is MPAA.
- Set the data type, the text used to prompt the user, whether to allow null, blank, multiple values or multiline text, and whether to hide the user interface.
- Select the default value or populate a list of available values from which users can choose.

Parameter values are collected in the order they appear in the Report Parameters collection. You can change the order using the arrows in the Report - Parameters dialog.

General Tab

- **Name**: Set the name for the parameter in this field. The value you supply here appears in the parameters list and must match the corresponding query parameter.
- **Data type**: Set the data type for your parameter which must match the data type of the field that it filters. The interface presented might also differ depending on the data type.
  - **Boolean**: Presents the user with two options True or False
  - **Date**: Presents the user with a calendar picker to select a date if you do not supply a default value or a drop-down selection of available values
  - **DateTime**: Presents the user with a calendar picker to select a date and a time picker to select the time in cases where you do not supply a default value or a drop-down selection of available values
  - **Integer**: Presents the user with a text box or a drop-down selection of available values
  - **Float**: Presents the user with a text box or a drop-down selection of available values
  - **String**: Presents the user with a text box or a drop-down selection of available values
- **Text for prompting users for a value**: Enter the text you want to see on the user interface to request information from the user in this field. By default this is the same as the Name property.
- **Allow null value**: Select this check box if you want to allow null values to be passed for the parameter. It is not selected by default.
- **Allow blank value**: Select this check box if you want to allow blank values to be passed for the parameter. It is not selected by default.
- **Multivalue**: Select this check box to allow the user to select multiple items in the available values list.
  For a multi-value parameter, you can also allow user to specify special value for 'Select all' option in the **Value for 'Select All'** property.
- **Multiline**: Select this check box to allow multiline values in the parameter. The control will automatically adjust to accommodate multiple lines.
- **Hidden**: Select this check box to hide the parameter interface from the user and instead provide a default value or pass in values from a subreport or drill-through link. Please note that if you hide the user interface and do not provide a default value, the report will not run.

Available Values

These values are used to fill a drop-down list from which the end user can choose.

- **Non-queried**: You can supply Labels and Values by typing in static values or using expressions.
- **From query**: You can select a Dataset from which to select a Value field and Label field.

Default Values

This is the value that you give for the parameter if the user does not supply one, or if you hide the parameter user interface.

- **Non-queried**: You can supply a default Value by entering a static value or using an expression.
- **From query**: You can select a Dataset from which to select a Value field.
- **None**: You can have your users provide a value for the parameter.

> **Note:** In the Available Values tab the **Value** is what is passed to the query parameter, and the **Label** is what is shown to the user. For example, if the Value is an Employee Number, you might want to supply a more user-friendly Label showing Employee Names.

To access the Report - Parameters Dialog

You can access the Report - Parameters dialog through any one of the following:

- In the Report Explorer, click the Add (+) icon and select the **Parameter** option.
- In the Report Explorer, right-click the Parameters node and select **Add Parameter**.
- In the Report Explorer, right-click the Report node and select **Report Parameters**.
- From the Report Menu, select **Report Parameters**.

The Report Parameters dialog contains a parameters page with a list of parameters and three tabs to set parameter properties. To add a parameter to the list, click the Add (+) icon and set the parameter properties in the three tabs described below.

Parameters page of the DataSet dialog

On the Parameters page of the DataSet Dialog, pass a Report Parameter into the parameter in your query. You can click the Add (+) icon at the top of the parameters list, enter parameter name, and supply a value like:

```
=Parameters!MPAA.Value
```

Query page of the DataSet dialog

On the Query page of the DataSet Dialog, enter the parameter in the SQL query. Use the syntax specific to your data source type to create a parameter. For example, with an OleDB data source, add a query like the following for a multi-value Movie Rating parameter:

```
SELECT * FROM Movie WHERE (MPAA = ? AND YearReleased = ?)
```

If you want to run a report without prompting the user for a value at run time, you need to set a default value for each parameter and the **Hidden** check box should be selected in the Report - Parameters dialog, General tab.

Subreport parameters are also considered as hidden parameters as a user can easily synchronize a subreport's data with that of the parent report. See Subreports in Page/RDL Reports for further details.

Drill-Through parameters are also hidden parameters as drill-through links are used to navigate from one report to another. When you select **Jump to report** for the action, the parameters list is enabled.

Section Report

In section report, you can use the Parameters collection to pass values directly into a control at run time, or you can also use it to display a subset of data in a particular instance of a report.

There are several ways for setting up parameters in a report:

- You can enter syntax like the following in your SQL query to filter the data displayed in a report at run time: `<%Name | PromptString | DefaultValue | DataType | PromptUser%>`
- You can add parameters through the Report Explorer and place them on the report as TextBox controls to pass values in them at run time.
- You can also add parameters through the code behind the report, inside the **ReportStart** event. See Add Parameters for more information.

Prompting for Parameter Values

In order to prompt the user for parameter values, all of the following must be in place:

- At least one parameter should exist in the Parameters collection of the report.
- The **PromptUser** property for at least one parameter must be set to **True**.
- On the report object, the **ShowParameterUI** property must be set to **True**.

When there are parameters in the collection and the ShowParameterUI property is set to True, the user prompt automatically displays when the report is run. When the user enters the requested values and clicks the **OK** button, the report gets displayed using the specified values.

Values of a parameter added through the Report Explorer can be applied to a parameter in the SQL query by specifying the `param:` prefix for a parameter in the SQL query. This prefix relates the current parameter to the one in the Report Explorer.

For e.g., `select * from CUSTOMERS where CustomerName = '<%param:Parameter1%>'`. In this case, the parameter with the `param:` prefix in the SQL query is updated with values of the corresponding parameter in the Report Explorer.

> **Note**: Within the same report, you can prompt users for some parameters and not for others by setting the **PromptUser** property to **True** on some and **False** on others. However, if the report object's **ShowParameterUI** property is set to **False**, the user prompt does not display for any parameters regardless of its **PromptUser** setting.

Adding Parameters to the Parameters Collection via the SQL Query

When you add a single parameter to a report's Parameters collection via the SQL query, the query looks like this:

SQL Query.

```
SELECT * FROM Products
INNER JOIN Categories ON Products.CategoryID = Categories.CategoryID
WHERE Products.SupplierID = <%SupplierID|Enter a Supplier ID|1|S|True%>
```

You can also create a parameterized query from the Visual Query Designer. See Query Building With Visual Query Designer for further information on how to create a parameterized query using the interactive query designer.

There are five values in the parameter syntax, separated by the pipe character: |

Only the first value (Name) is required, but if you do not specify the third value (DefaultValue), the field list is not populated at design time. You can provide only the **Name** value and no pipes, or if you wish to provide some, but not all of the values, simply provide pipes with no space between them for the missing values. For example, <%ProductID||||False%>

Name: This is the unique name of the parameter, and corresponds to the Key property in parameters entered via code.
PromptString: This string is displayed in the user prompt to let the user know what sort of value to enter.
DefaultValue: Providing a default value to use for the parameter allows ActiveReports to populate the bound fields list while you are designing your report, enabling you to drag fields onto the report. It also populates the user prompt so that the user can simply click the **OK** button to accept the default value.
DataType: This value, which defaults to S for string, tells ActiveReports what type of data the parameter represents. It also dictates the type of control used in the user prompt. The type can be one of three values.

- **S (string)** provides a textbox into which the user can enter the string.
  Depending on your data source, you may need to put apostrophes (single quotes) or quotation marks around the parameter syntax for string values.
  For example, `'<%MyStringParameter%>'`
  Also, if you provide a default value for a string parameter that is enclosed in apostrophes or quotation marks, ActiveReports sends the apostrophes or quotation marks along with the string to SQL.
  For example, `<%MyStringParameter||"DefaultValue"|S|False%>`
- **D (date)** provides a drop-down calendar picker from which the user can select a date.
  Depending on your data source, you may need to put number signs around the parameter syntax.
  For example, `#<%MyDateParameter%>#`
- **B (Boolean)** provides a checkbox which the user can select or clear.
  If you provide a default value of True or False, or 0 or 1 for a Boolean parameter, ActiveReports sends it to SQL in that format.

  > **Note**: In case of Microsoft Access Database, the default value for boolean parameter is specified as -1(true) or 0(false).

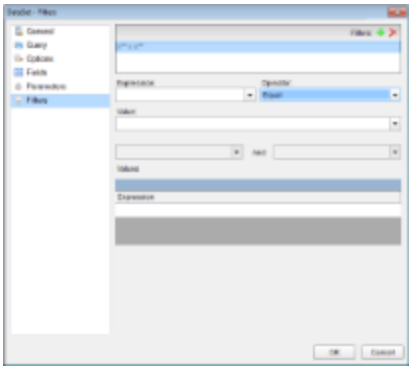
PromptUser: This Boolean allows you to tell ActiveReports whether to prompt the user for a value. This can be set to True for some parameters and False for others. If you set the report's ShowParameterUI property to False, users are not prompted for any parameters, regardless of the PromptUser value set for any parameter in the report.

Filtering

In page layout, ActiveReports allows you to set filters on a large set of data that has already been retrieved from the data source and use them with datasets or data regions to limit the information you want to display on your report.

Although not as efficient performance-wise as query parameters which filter data at the source, there are still scenarios which demand filters. The obvious case is when the data source does not support query parameters. Another case for using filters is when users who require different sets of data view the same report.

You can set filters on a **Filters** page or a tab similar to the one in the following image.



There are three major elements that constitute a filter:

- **Expression:** Type or use the expression editor to provide the expression on which to filter data.
- **Operator:** Select the operator to compare the expression results with the Value.
- **Value:** Enter the value with which to compare the expression results.

For example, in the filter `=Fields!YearReleased.Value = 1997` applied on a dataset from the Movies table of the Reels.mdb database, **=Fields!YearReleased.Value** is set under expression, **=** is the operator and **1997** is the value on which filter is set. See Set Filters for further instructions on adding filters in reports.

You can also use multiple values with the **In** and **Between** operators. Two fields with an *And* in the middle appear for the Between operator, and another Expression field is available at the bottom of the Filters page or tab for the In operator. The following table lists all available filtering operators.

Filtering Operators

Filter	Description
Equal	Select this operator if you want to choose data for which the value on the left is equal to the value on the right.
Like	Select this operator if you want to choose data for which the value on the left is similar to the value on the right. See the MSDN Web site for more information on the Like operator.
NotEqual	Select this operator if you want to choose data for which the value on the left is not equal to the value on the right.
GreaterThan	Select this operator if you want to choose data for which the value on the left is greater than the value on the right.
GreaterThanOrEqual	Select this operator if you want to choose data for which the value on the left is greater than or equal to the value on the right.
LessThan	Select this operator if you want to choose data for which the value on the left is less than the

	value on the right.
LessThanOrEqual	Select this operator if you want to choose data for which the value on the left is less than or equal to the value on the right.
TopN	Select this operator if you want to choose items from the value on the left which are the top number specified in the value on the right.
BottomN	Select this operator if you want to choose items from the value on the left which are the bottom number specified in the value on the right.
TopPercent	Select this operator if you want to choose items from the value on the left which are the top percent specified in the value on the right.
BottomPercent	Select this operator if you want to choose items from the value on the left which are the bottom percent specified in the value on the right.
In	Select this operator if you want to choose items from the value on the left which are in the array of values on the right. This operator enables the Values list at the bottom of the Filters page.
Between	Select this operator if you want to choose items from the value on the left which fall between pair of values you specify on the right. This operator enables two Value boxes instead of one.

Drill-Down Reports

The drill-down feature helps in temporarily hiding a part of your report. That hidden part can be controls, groups, columns or rows. When you open a drill-down report, part of the data is hidden so that you can only see high-level data until you request for more detail. In such reports you find an expand icon (plus-sign image) next to the toggle item in the report. Clicking the toggle image, or plus sign, expands hidden content into view and the expand icon changes to a collapse icon (minus-sign). When you click the minus-sign image, it hides the content and returns the report to its previous state.

To create a drill-down report, use the **Visibility** properties of controls, groups, columns, or rows. Simply set the Visibility-hidden property to **True** and set the toggle item to the name of another item in the report, usually a text box in the group containing the hidden item. At run time, this puts a plus sign next to the toggle item which the user can click to display the hidden data.

If you export a drill-down report or render it through rendering extensions, any content which is hidden at the time of export remains hidden in the exported file. If you want all of the content to appear in the exported file, you must first expand all hidden data.

Only when you render a report using the XML using the **XmlRenderingExtension ('XmlRenderingExtension Class' in the on-line documentation)**, all hidden data is exported regardless of whether it is hidden at the time of export.

Linking in Reports

You can enhance the interactivity in your report by adding different types of links to it. ActiveReports provides the ability to add bookmarks, hyperlinks, drill-through links to reports.

The following topic explains the links you can create in page and section reports.

Page Report/RDL Report

Hyperlinks

Hyperlinks take you to a web page that opens in the default browser of the system. You can set hyperlinks in the Textbox, Image, Chart, and Map controls to access a Web page from your report. See Add Hyperlinks for further information.

Hyperlinks are displayed when you preview a page report or a RDL report in the Viewer, export a report in HTML , PDF, RTF, and Excel formats. You can also see hyperlinks in Word, HTML, and PDF formats when you render reports using rendering extensions.

Bookmarks

A bookmark link is similar to a hyperlink, except that it moves the viewer to another area in the report instead of linking to a web page. You can create these links on a control using a Bookmark ID that connects to another target control. See Adding Bookmarks for further information.

Bookmarks are displayed when you preview a page report/RDL report in the Viewer or render a report through rendering extensions in Word, HTML, and PDF formats.

Drill through links

A drill-through link takes you to another report with more detail. Drill-through links appear as a hyperlink that you can click to move to a completely different report. You can also create more complex links where you pass parameters to the report called by the link.

Drill-through links are displayed when you preview a page report/RDL report in the Viewer or render a report through rendering extensions in HTML format.

> **Note**: While rendering a report to HTML, drill-through links are broken unless the target report is also exported to the same directory with the same name as the original.

Section Layout

Hyperlinks

Hyperlinks take you to a web page that opens in the default browser of the system. You can set the HyperLink property available with the Label, Textbox, Picture and OleObject controls that allow you to add hyperlinks that connect to a Web page. You can also use the HyperLink property to open an e-mail or jump to a bookmark. See Add Hyperlinks for further details.

Hyperlinks are supported when you preview a section report in the Viewer, export a report in HTML , PDF, RTF and Excel formats.

Bookmarks

Bookmark links take you to a location where the bookmark is set on your report. Unlike hyperlinks, these links take you within the report. Bookmarks and nested bookmarks appear in the document map for fields, groups and subreports. You can also add special bookmarks at run-time. You can use hyperlinks for simulating a drill through like feature similar to Page Layout. See Add Bookmarks for further details.

Bookmarks are supported when you preview a section report in the Viewer, export a report in HTML and PDF formats.

> **Note**: When you export a report to HTML, a*.TOC file is created in the folder where you export the report. Use this file to reach the bookmarked locations.

Document Map

When you click an item in the document map, the viewer jumps to that item in the report.

A document map functions as a table of contents for a page report or a RDL report and as a bookmarks panel for a section report. It provides a convenient way to navigate a lengthy report.

Page Reports/RDL Reports

In a page report or a RDL Report, you can add report controls, data regions, groups and detail groups to the document map by:

- Assigning a value to the **Document map label** on the Navigation page of the corresponding dialog.
- Setting the value of the **Label** property in the properties window.
- Setting the value of the **HeadingLevel** property in the properties window.

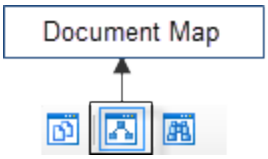
See Add Items to the Document Map for more information.

Section Reports

In a section report, when you add a bookmark on any control it appears in the document map while viewing the report. In order to navigate to a bookmark you need to open the document map and click that bookmark.

See Add Bookmarks for more information.

Viewing the Document Map in the Viewer

1. On the Viewer toolbar, click the Toggle sidebar button to display the sidebar.
   
2. At the bottom of the sidebar pane, click the **Document map** button to display the document map.
   

   If there is no document map associated with the report, the button does not appear at the bottom of the sidebar pane.
3. In the Document map that appears, click the item you want to view in the report.

> **Note:** You can also access the Document map from the Toggle sidebar button on the preview tab toolbar.

Exporting document maps

In the Viewer, the document map appears in a sidebar to the left of the report, but when you export your page or section report to various file formats, they handle document maps differently.

Export Filter	Effect on Document Map
HTML	A .toc file containing the document map is exported along with the HTML report.
PDF	Document map appears in the bookmarks panel.
Text	Document map does not appear in the exported report.
Rich Text Format	Document map does not appear in the exported report.
TIFF	Document map does not appear in the exported report.
Excel	Document map does not appear in the exported report.

In a page report or a RDL report, if you use rendering extensions to export your report, the document map is not available in any rendering type except PDF where it appears in the bookmarks panel.

Note: For printing and rendering purposes use Table of Contents control in your page report and RDL report.

Sorting

In order to better organize and present data in your report, you can sort it alphabetically or numerically in ascending or descending order. You can also use sorting effectively with grouped data to present an easy to understand, comprehensive view of report data.

Sorting in Page Reports/RDL Reports

In a page report or a RDL report, you can sort data in the data region, along with grouping, on a fixed page in Page report or sort data directly in the SQL query. You can also set interactive sorting for your data on a TextBox control.

Sorting at different levels in a Report

You can apply sorting at different levels on your report data. ActiveReports provides a Sorting page in the dialogs of a data region, grouped data and fixed page to determine where you want to display sorted data.

Sorting data in a Data Region

In Table and List data regions, you can sort data within the data region. To sort data within these data regions, set sorting in the **Sorting** page of the specific data region's dialog.

In Tablix, BandedList and Chart data regions, sorting is only possible on grouped data therefore there is no independent Sorting page available in their specific dialogs.

Sorting grouped data

A Sorting tab is available inside the Groups page of all the data region dialogs and the Detail Grouping page of the List dialog. It allows you to set the sort order of grouped data. This tab is enabled once grouping is set inside the data region.

Sorting on a Fixed Page

In a Page report, sorting is also possible on a fixed page grouped on a dynamic value. Sorting data on a fixed page is similar to sorting grouped data in a data region. The only difference is when you sort data on the fixed page you apply sorting to all the data regions that are placed on the design surface. See, Sort Data for more information.

Sorting data through SQL Query

When you connect to a data source and create a data set to fetch data for your report, you define a query. Set the ORDER

BY keyword in the query to sort data in ascending or descending order.

By default, the ORDER BY keyword usually sorts the data in ascending order, but you can include the DESC keyword in your query to sort data in descending order. For example, if you want to fetch data from the *Movie* table of the Reels database and sort it on the *Title* field, your query looks like the following:

```
SELECT * FROM Movie ORDER BY Title
```

OR

```
SELECT * FROM Movie ORDER BY Title ASC
```

In case you want the *Title* field sorted in descending order, your query looks like the following:

```
SELECT * FROM Movie ORDER BY Title DESC
```

Interactive Sorting

You can add interactive sorting on a TextBox control to allow users to sort columns of data within a data region on a published report.

The interactive sorting feature is set through the **Interactive Sort** page which available in the TextBox dialog.

Once you set interactive sorting on a TextBox control, while viewing the report in the Viewer or in the Preview Tab the textbox control displays a sort icon inside it. A user can sort data that appears inside the textbox in ascending or descending order by clicking the icons.

On the Interactive Sort page of the TextBox dialog you can find following fields available for entering values:

- **Sort Expression:** An expression specifying the sort value for data contained in the column.
- **Data region or group to sort:** Select the grouping level or data region within the report to sort. The default value is Current scope, but you may also choose an alternate data region or grouping.
- **Evaluate sort expression in this scope:** Select the grouping level within the report on which to evaluate an aggregate sorting expression. The default value is Current scope, but you may also choose an alternate data region or grouping.

See Allow Users to Sort Data in the Viewer for more information.

Sorting in Section Reports

In a section report, sorting is not explicitly available. However, you can modify the SQL Query to order your data while fetching it from the database.

Sorting data through SQL Query

When you connect the report to a data source and enter a query to fetch data, you can include the ORDER BY keyword in your query to get sorted data.

By default, the ORDER BY keyword usually sorts data in ascending order, but you can include the DESC keyword in your query to sort data in descending order. For example, if you want to fetch data from the *Customers* table of the NWind database and sort it on the *CompanyName* field, your query looks like the following:

```
SELECT * FROM Customers ORDER BY CompanyName
```

OR

```
SELECT * FROM Customers ORDER BY CompanyName ASC
```

In case you want the *CompanyName* field sorted in descending order, your query looks like the following:

```
SELECT * FROM Customers ORDER BY CompanyName DESC
```

Annotations

Annotations are floating text bars or images to call attention to specific items or values or to add notes and special instructions directly to the reports. Annotations added via the viewer's toolbar are temporary and are destroyed when the report closes.

These annotations are accessible through the **Annotation** button present on the Viewer toolbar which is hidden by default. You can make the Annotations toolbar visible by setting the **AnnotationDropDownVisible ('AnnotationDropDownVisible Property' in the on-line documentation)** property to True in the viewer's properties window.

Available Annotations

Each annotation type allows you to change the colors, transparency, border, font, and alignment, plus other properties specific to the type of annotation. Available annotations include:

Annotation Name	Description
AnnotationText 	A rectangular box to enter text using the **Text** property.
AnnotationCircle 	A circle without text. You can change the shape to an oval by dragging its corners.
AnnotationRectangle 	A rectangular box without text. You can change the shape to a square by dragging its corners.
AnnotationArrow 	A 2D arrow to enter text using the **Text** property. You can also change the arrow direction using the **ArrowDirection** property.
AnnotationBalloon 	A balloon caption to enter text using its **Text** property. You can also point the balloon's tail in any direction using its **Quadrant** property.
AnnotationLine 	A line to enter text above or below it using its **Text** and **LineLocation** properties. You can also add arrow caps to one or both ends and select different dash styles using **DashCap**, **DashStyle**, and **ShowArrowCaps** properties.
AnnotationImage 	A rectangle with a background image and text. You can select an any image inside it using the **BackgroundImage** property. You can also place text on the image using the **Text** property.

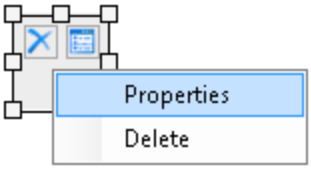
To add annotations using the Viewer

These steps assume that you have already placed the Viewer control onto a Windows Form and loaded a report in it. See Windows Forms Viewer for more information.

1. In your Visual Studio project, on the Form where the Viewer control is placed, select the Viewer control an right-click to choose Properties.

2. In the Properties Window that appears, set the AnnotationDropDownVisible property to **True** to get an additional toolbar in the viewer control.

3. Run the report application and select the annotation you want use from the Annotation toolbar on the Viewer.
4. Drag the annotation to the desired location on the report design surface. The annotation appears with a **Delete** and a **Properties** button on the top left corner.

5. Inside the annotation, click the Properties button to view its properties in the Properties Window and use those properties to enter text, change color or transparency, set border or font, alignment etc.
6. Close the Properties Window to apply changes to the annotation.
7. Drag the corners to resize the annotation as needed. You can also select entire annotation to move it to another location on the report.
8. Right-click the annotation to display the annotation context menu. The context menu includes the **Properties** and **Delete** commands.


> **Note**: You can print a page, RDL or section report that contains annotations. In a section report, you can also save a report with annotations in RDF format. See Add and Save Annotations for further details.

Report Parts

What are Report Parts?

Report parts are groups of controls (with data and settings) in a report that you can reuse in other reports. Report parts are supported in Page, RDL, and Section reports. For example, you can use Chart and Tablix data regions from Report1, and a Table data region from Report2 to create a new report.

> **Note:** The Report Parts feature is only available with the Professional Edition license.

Why use report parts?

Enhanced reusability: With Report Parts, you can reuse report controls along with their associated data resources such as data sets and data source connections. In earlier versions, you could copy controls from one report to another, but not the resources associated with them. Using report parts, you can add everything that is required for that control to work.

Automatic conflict resolution: ActiveReports automatically resolves name conflicts for report parts. For example, if a report already contains a Tablix1 data region and you add a report part named Tablix1, it automatically changes the new report part's name to Tablix2.

Zero dependency: After adding a report part to a report, you can modify it independent of the original report part. For example, if you add Table1 as a report part in your current report, you can modify its properties without impacting the Table1 settings in your original report.

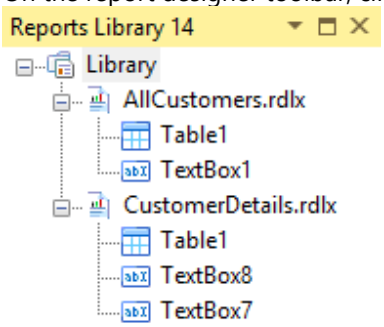
Here is some guidance on how to work with Report Parts

- **Show or hide the Reports Library**
- **Add report parts from local reports**
- **Hide report parts from the Reports Library**
- **Clear report parts from the Reports Library**
- **Use report parts in your reports**
- **Limitations**

Show or Hide the Reports Library

When ActiveReports is installed on your system, a **View Reports Library** button is automatically added to the Visual Studio toolbar. It appears every time you create a new application.

1. Right-click the Visual Studio toolbar and select **ActiveReports 14** to display the report designer toolbar. See Toolbar for further details.
2. On the report designer toolbar, click the **View Reports Library** button. The **Reports Library 14** window appears.



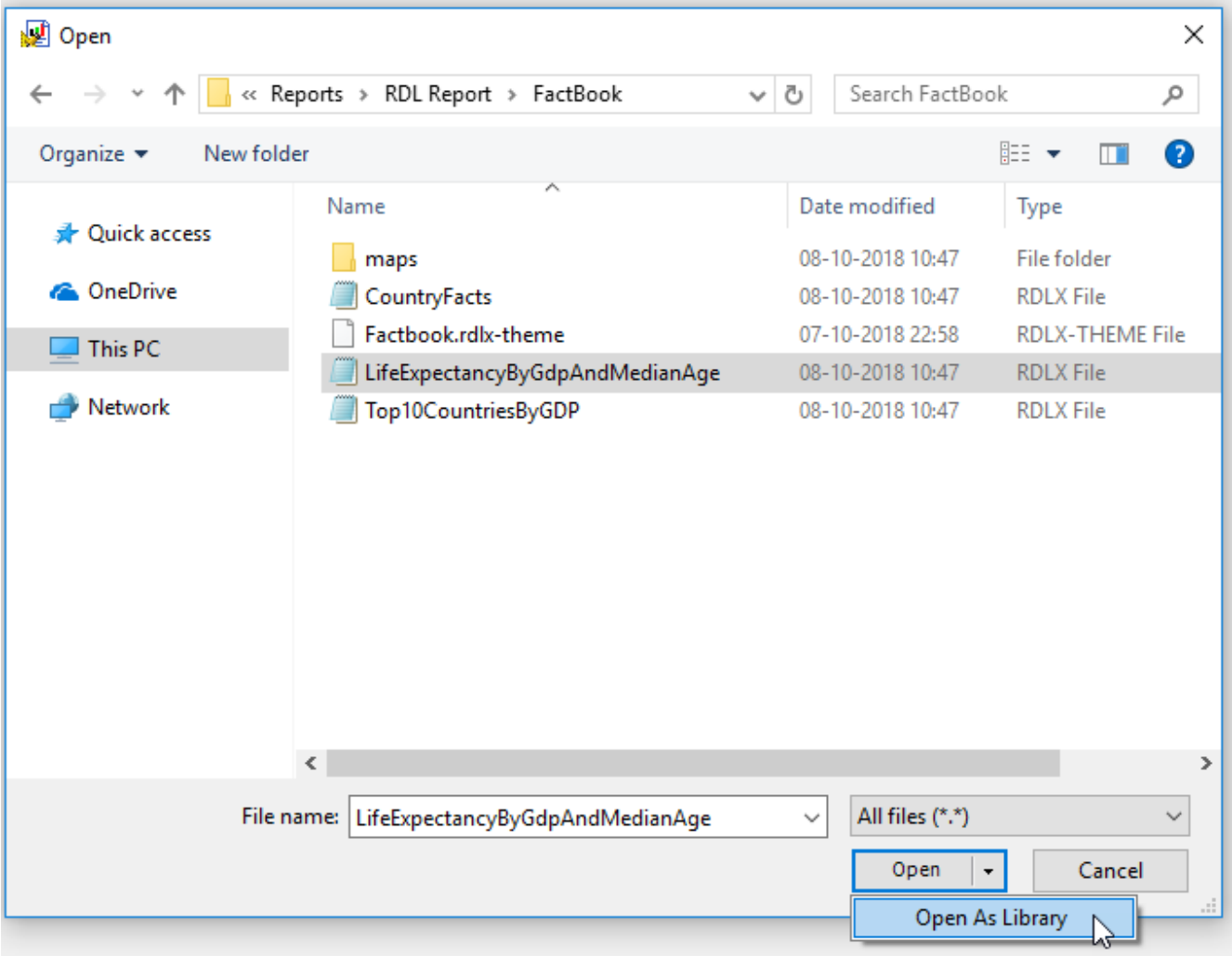
3. Click the View Reports Library button again to hide the **Reports Library 14** window.

Note:

- If the Reports Library window does not appear automatically in your application, select **View > Other Windows > Reports Library 14** in Visual Studio.
- The stand-alone designer application (GrapeCity.ActiveReports.Designer.exe) also contains a Reports Library window. See Standalone ActiveReports Designer for more information.
- Report authors can place reports in the **GrapeCity Reports Library** folder to show them as report parts in the Reports Library window. By default, the GrapeCity Reports Library folder is located in the User Documents folder.

Add report parts

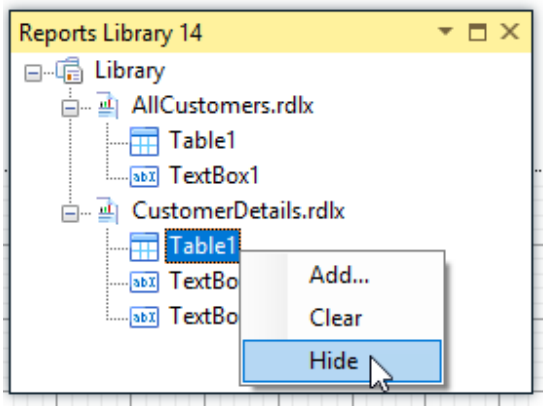
1. In the **stand-alone designer**, click the **File** menu and select **Open**. Alternatively, to add a report part from the **Reports Library** window, right-click the Reports Library node and select **Add**.
2. In the **Open** dialog that appears, navigate through the folder hierarchy and select the report you want to use for report parts.



3. Click the dropdown arrow next to the **Open** button and select **Open As Library** to add report parts to the **Reports Library 14** window. Once the report parts are added to your Report Library, you can use them to design new or existing reports.

Hide report parts from the Reports Library

1. Open the **Reports Library** window. For more information on how to open Reports Library window, see **Show or Hide the Reports Library**
2. In the **Reports Library** window, select the report part that you want to hide.



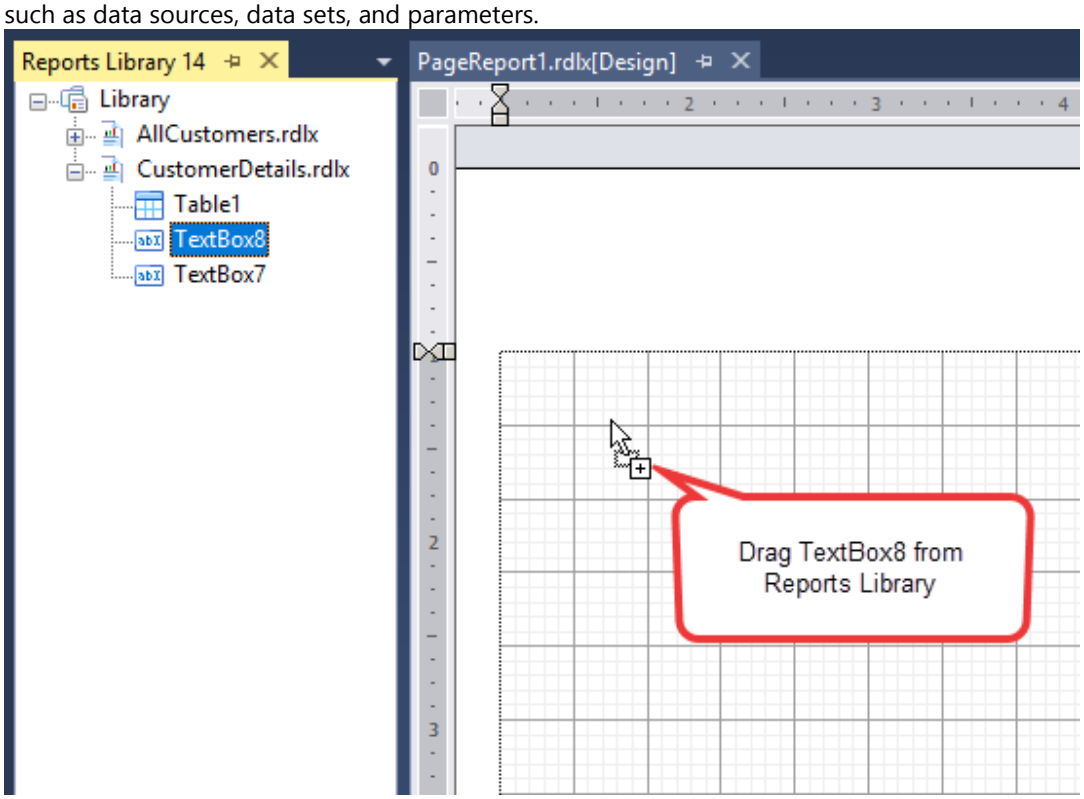
3. Right-click on the selected report part and then select **Hide** option from the context menu to hide the selected report part from Reports Library window.

Clear report parts from the Reports Library

1. Right-click anywhere inside the **Reports Library** window and select the **Clear** option from the context menu to completely clear the Reports Library window.

Use report parts in your reports

1. Once you have added the Report Parts to the **Reports Library** window, you can drag and drop the control from the Reports Library window onto the design surface. The report part is added to your report along with its dependencies such as data sources, data sets, and parameters.



Note: If you are working with a Page or RDL report, Section report parts are disabled in the Reports Library

> window and vice versa.

Limitation

- Page and RDL reports: CheckBox, Shape, Line, Subreport, OverflowPlaceHolder and ContentPlaceHolder controls are not supported as report parts.
- Section reports: Picture, RichTextBox, Barcode, and Chart are the only controls that you can add as report parts. The rest of the controls are not supported as report parts.

Create report using Report Parts

This topic illustrates a step-by-step implementation for creating a report using Report Parts to display annual sales of an organization.

> **Note**:
>
> - This topic uses the SalesReport.rdlx report file. The SalesReport.rdlx file can be downloaded from GitHub: ..\Samples14\Data\Reels.mdb.
> - Although this topic uses Page reports, you can also implement report parts in RDL and Section reports.

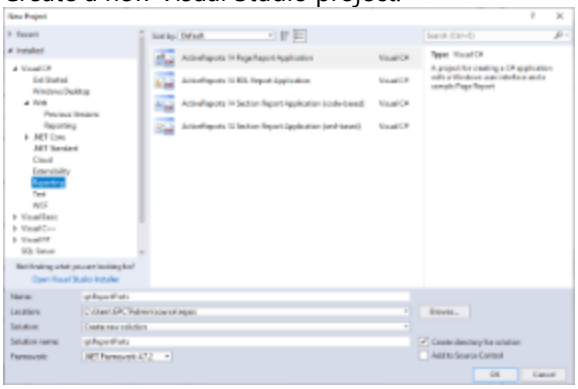
When you complete these steps, you will have a layout that looks similar to the following at design time and at run time.

Design Time Layout

Run-Time Layout

Create an ActiveReports project in Visual Studio

1. Create a new Visual Studio project.



2. In the **New Project** dialog that appears, select **ActiveReports 14 Page Report Application** and in the Name field, name the file as rptReportParts.
3. Click **OK** to create a new **ActiveReports 14 Page Report Application**. By default a Page report is added to the project.

See Quick Start for information on adding different report layouts.

Create a layout for the report

1. Select **PageReport1.rdlx** from the Solution Explorer.
2. Open the **Reports Library** window. For more information on how to open the Reports Library window, see Show or Hide the Reports Library.
3. To add a report part from the **Reports Library** window, right-click the Library node and select **Add**.
4. In the **Open** dialog that appears, navigate to **SalesReport.rdlx** file. Controls from SalesReport.rdlx are added as Report parts to your Report Library 14 window.
5. Drag and drop the **salesOverTime** chart control from the Reports Library window onto the design surface. The report part is added to your report along with its data sources, data sets and parameters.
6. Drag and drop the **Image1** image from the Reports Library window onto the design surface.
7. From the Visual Studio toolbox, drag and drop a **Table** data region onto the design surface.
8. Hover over TextBox4 to reveal the field selection adorner, click it to display a list of available fields, and select the **SaleDate** field.
9. Hover over TextBox5 to reveal the field selection adorner, click it to display a list of available fields, and select the

Quantity field.

10. Hover over TextBox6 to reveal the field selection adorner, click it to display a list of available fields, and select the **Profit** field.
11. From the Visual Studio toolbox, drag and drop a **Textbox** control onto the design surface.
12. Select the Textbox to view its properties in the Properties window and enter the text **Annual Sales Report (2004-2005)** in the Value property.

Enhance the appearance of the report

When you preview the report at this point, you will notice the data from the fields is displayed in the Table data region. Let us now work on the report appearance to further enhance the layout.

1. From the designer, select the **Image** control and then go to Properties Window to set the following properties.

Property Name	Property Value
Location	0.095in, 0.156in
Size	1.3in, 1.1in

2. From the designer, select **TextBox10** and then go to Properties Window to set the following properties.

Property Name	Property Value
Location	1.5in, 0.5in
Size	4.3in, 0.3in
BorderStyle	Solid
FontWeight	Bold
TextAlign	Center
FontSize	16pt

3. From the designer, select the **Chart** data region and then go to Properties Window to set the following properties.

Property Name	Property Value
Location	0.1in, 1.3in
Size	6in, 4.125in

4. From the designer, select the **Table** data region and then go to Properties Window to set the following properties.

Property Name	Property Value
Location	1.25in, 5.5in
Size	4.25in, 1in
FixedSize	4.25in, 3.35in
RepeatHeaderOnNewPage	True

5. In the Table data region, select the textbox that contains the following fields and go to the Properties window to set the following properties.

Cell	Property Name	Property Value
Sale Date	BackgroundColor	LightCyan
	BorderStyle	Solid

	FontWeight	Bold
	TextAlign	Center
=[SaleDate]	BorderStyle	Solid
	TextAlign	Center
Quantity	BackgroundColor	LightCyan
	BorderStyle	Solid
	FontWeight	Bold
	TextAlign	Center
=[Quantity]	BorderStyle	Solid
	TextAlign	Center
Profit	BackgroundColor	LightCyan
	BorderStyle	Solid
	FontWeight	Bold
	TextAlign	Center
=[Profit]	BorderStyle	Solid
	TextAlign	Center
	Format	Currency

View the report

- Click the preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

Common Concepts

Some common concepts related to text appearance in report controls are discussed in following topics:

- Text Justification
- Multi Line in Report Controls
- Line Spacing and Character Spacing
- Shrink Text to Fit in a Control
- Condense Characters to Fit in a Control

Text Justification

The **TextJustify Property (on-line documentation)** of a Textbox control provides you justification options for aligning

your text within a control. It is important that the **TextAlign** property (**Alignment** property in a Section Report) must be set to Justify' for **TextJustify** property to affect the text layout.

> **Note**: In section layout, the TextJustify property is also available in the Label control.

You can choose from the following values of the TextJustify property:

Auto

Results in Standard MSWord like justification where space at the end of a line is spread across other words in that line. This is the default value.

Distribute

Spaces individual characters within a word, except for the last line.

DistributeAllLines

Spaces individual characters within words and also justifies the last line according to the length of others lines.

To set Text Justification

1. On design surface, select the control to view it's properties in Properties window.
2. In the properties window, set the **TextAlign** property (Alignment property in a Section Report) to **Justify**.
3. Go to the **TextJustify** property and from the drop down list select any one option.

Text justification is supported when you preview a report in the Viewer, print a report or export a page, RDL or section report in PDF and TIFF formats. In page reports and RDL reports, it is also supported while rendering a report in Word, HTML, PDF and Image formats using rendering extensions. See Rendering Extensions for more information on rendering extensions.

Multi Line in Report Controls

ActiveReports allows you to display text within a control on multiple lines.

Multiline in Section Reports

In a section report, to enable multiline display in your report control, you need to set the **Multiline Property (on-line documentation)** to True. Then, with your control in edit mode, insert line breaks at the desired location using the **Enter** key or **Ctrl** + **Enter** keys to create multiline text. However, when the MultiLine property is set to False, text entered into the control is displayed on a single line.

You can display multiline text in TextBox, RichTextBox and Label controls.

Multiline in Page Reports/RDL Reports

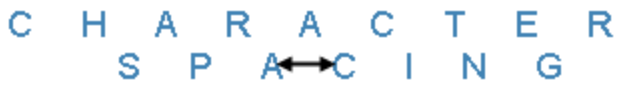
In a page report or a RDL Report, with your control in edit mode, insert line breaks at the desired location using the **Enter** key or **Ctrl** + **Enter** key to create multiline text. You can also insert line breaks in the Expression Editor through the Value property of the control.

You can display multiline text in the TextBox and CheckBox controls.

> **Note:** In edit mode, scrollbars appear automatically to fit multiline content within a control. However, these are not displayed in the preview tab, so you may need to adjust the **Size** property of the control to display all of the text.

Line Spacing and Character Spacing

In ActiveReports, in order to make your report output clearly visible during export or printing you can set character and line spacing. In order to use line spacing you must first set the MultiLine property for the control to True.



The **CharacterSpacing ('CharacterSpacing Property' in the on-line documentation)** and **LineSpacing ('LineSpacing Property' in the on-line documentation)** properties are available in the following controls for this purpose:

Page Report/RDL Report

- TextBox

Section Report

- TextBox
- Label

To set line or character spacing

1. On the design surface, click the control to display it in the Properties Window.
2. In the Properties window, click the **Property Dialog** command at the bottom to open the control dialog.
3. In the TextBox dialog, go to the **Format** page and set the Line Spacing or Character Spacing values in points.

> **Note**: You can also set the CharacterSpacing and LineSpacing property directly in the Properties Window.

Line and character spacing is supported when you preview a report in the Viewer, print a report or export a section report in HTML , PDF and TIFF formats.

In page reports and RDL reports, it is also supported while rendering a report through rendering extensions in Word, HTML, PDF and Image formats. See Rendering Extensions for further information on rendering extensions.

Shrink Text to Fit in a Control

In ActiveReports, when working with the Textbox control in a page report and RDL report or a TextBox or Label control in a section report, you can use the **GrapeCity.ActiveReports.PageReportModel.Style.ShrinkToFit** property to reduce the size of the text so that it fits within the bounds of the control. The text shrinks at run time, so you can see the reduced font

size when you preview, print or export the report.

The following image illustrates the result when the **ShrinkToFit** property is set to True on Title.

Title	Year Release	User Rating
[illegible]	1997	9.1
Star Wars	1977	8
Shrek 2	2004	7.1
[illegible]	1982	7.3
[illegible]	1999	9.1
Spider-Man	2002	8.6
[illegible]	2005	8.5
[illegible]	2003	8.5
Spider-Man 2	2004	8.8
[illegible]	2004	8.8
Jurassic Park	1993	8.8
[illegible]	2002	8.9
Finding Nemo	2003	7.2
Forrest Gump	1994	8.4

You can use other text formatting properties in combination with the **ShrinkToFit** property.

> ⚠ **Caution**:
>
> - When both **CanGrow** and **ShrinkToFit** are set to True, CanGrow setting is ignored and only ShrinkToFit is applied to the report.
> - When **ShrinkToFit** is set to True and **Angle** is set to a value other than 0, the ShrinkToFit property is ignored.
> - **ShrinkToFit** property does not work in following conditions:
>   - VerticalText property (in Section Report) or WritingMode property (in Page/RDL report) is set for a control.
>   - Multiline property in Section report is set to False.
> - Common value with Page number (in data region or report header/footer) appears clipped in exported files (HTML, MHT, Word, and Excel) even when ShrinkToFit property for the TextBox is set to True.

Export Support

While exporting a report, various file formats handle **ShrinkToFit** differently.

ShrinkToFit gets exported in all formats except Text. While rendering a page report or RDL report using rendering extensions, ShrinkToFit is not supported in XML. However, all other rendering extensions allow ShrinkToFit display as it is. See Rendering Extensions for more on rendering extensions.

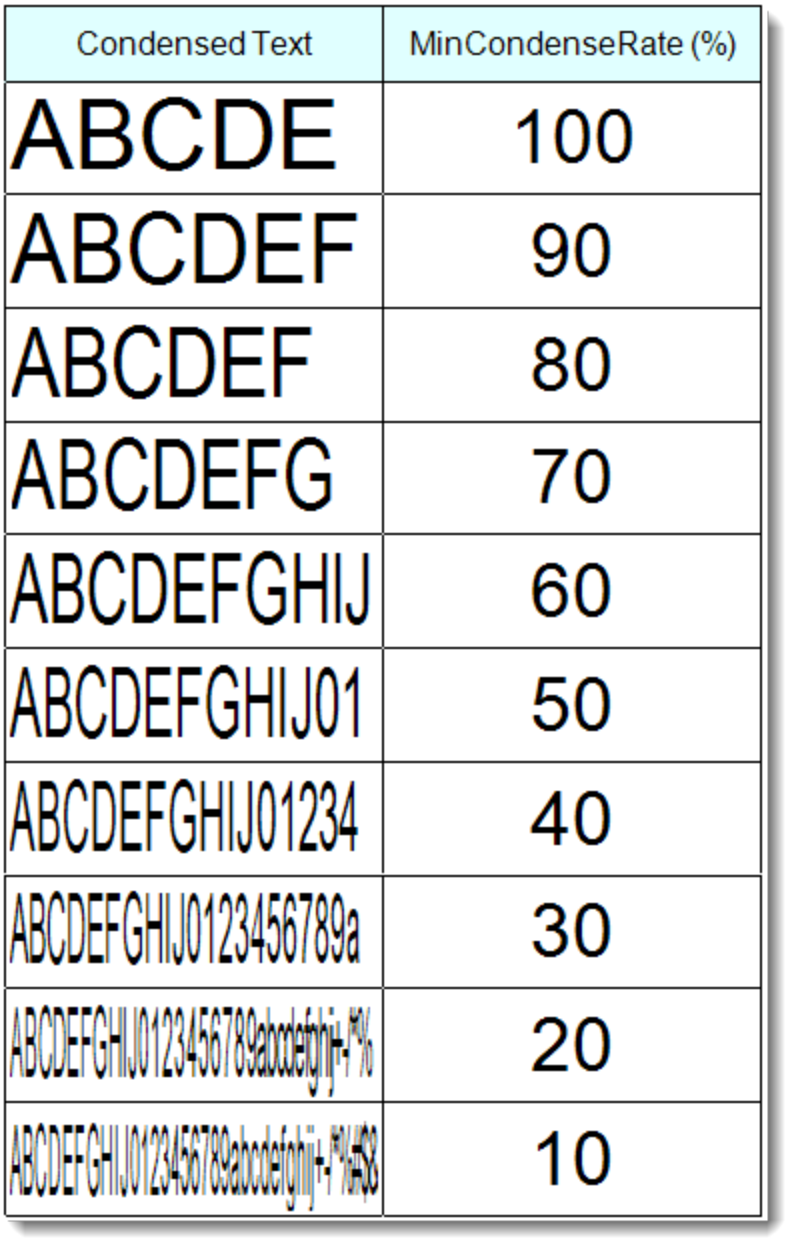
Condense Characters to Fit in a Control

The **MinCondenseRate ('MinCondenseRate Property' in the on-line documentation)** property for Label and TextBox controls in Section and Page/RDL reports allows users to shrink the width of the text so that it fits within the width of the control. With MinCondenseRate property, the size of the text is compressed horizontally and all characters and character-spacing in the text shrink with the same ratio; and with ShrinkToFit property, both height and width of a text are compressed.

When the text is multi-line, the width of the text shrinks based on the longest line. If the text cannot fit the width of the control even after the width of the text is compressed with the minimum ratio, line breaks are inserted in the middle of the text depending on the setting of the WrapMode property.

The default value of the MinCondenseRate property is 100 (percentage) and the available range is from 10 to 100 (only natural number).

The following image illustrates how the text "ABCDEFGHIJ0123456789abcdefghij+-/*%#$&" is compressed when the **MinCondenseRate** property for a TextBox is set from 10 to 100.



Condensed Text	MinCondenseRate (%)
ABCDE	100
ABCDEF	90
ABCDEF	80
ABCDEFG	70
ABCDEFGHIJ	60
ABCDEFGHIJ01	50
ABCDEFGHIJ01234	40
ABCDEFGHIJ0123456789a	30
ABCDEFGHIJ0123456789abcdefghij+-!%	20
ABCDEFGHIJ0123456789abcdefghij+-!%#$8	10

⚠ **Caution**: When **MinCondenseRate** property is specified, the following properties are ignored:

- Angle (Label control in Section report and TextBox control in Page/RDL report)
- ShrinkToFit
- VerticalText (Label control in Section report, and TextBox control in Section and Page/RDL reports)
- WritingMode (TextBox control in Page/RDL report)

Export Support

The Condensed character feature is supported in PDF, HTML, and MHT export formats when exporting Page/RDL reports using Rendering Extensions.

⚠ **Caution**: The Condensed characters are NOT supported when exporting Page/RDL reports using PDF and HTML

> Export Filters.

Viewer Support

The Condensed character feature is supported in the WinForms, WPF, and Web viewers.

> ⚠

Localization

ActiveReports uses the Hub and Spoke model for localizing resources. The hub is the main executing assembly and the spokes are the satellite DLLs that contain localized resources for the application.

For example, if you want to localize the Viewer Control, the hub is GrapeCity.ActiveReports.Viewer.Win.dll and the spoke is GrapeCity.ActiveReports.Viewer.Win.resources.dll.

In your Program Files folder, the Localization folder is in a path like **....\GrapeCity\ActiveReports 14\Localization**, and contains all of the ActiveReports components that you can localize.

There are 16 ActiveReports components in the Localization folder and most have two files.

- A **.bat** file that is used to set the Cultures to which you want to localize.
- A **.zip** file that contains the resource files (**.resx**) in which you update or change the strings.

There is one application in the Localization folder: **NameCompleter.exe**. When you run your .bat file after updating your culture, it runs this application to create a SatelliteAssembly folder with a language sub-folder containing the localized GrapeCity.ActiveReports.*AssemblyName*.resources.dll file.

Place the language folder containing the *.resources.dll file inside your main executing assembly folder to implement changes.

> **Note**: Before you can distribute or put your localization in the Global Assembly Cache (GAC), you must first send the localized GrapeCity.ActiveReports.*AssemblyName*.resources.dll file to Support Team by opening a support ticket, and get it signed with a strong name. Once you receive the signed DLL file, you can drag the language subfolder with the signed DLL file into C:\WINDOWS\ASSEMBLY, or distribute it with your solution.

When the main executing assembly needs a resource, it uses a ResourceManager object to load the required resource. The ResourceManager uses the thread's **CurrentUICulture** property.

The common language run time sets the CurrentUICulture property or you can set it in code to force a certain UI Culture so that you can test whether your satellite DLL is loading properly. The ResourceManager class uses the CurrentUICulture property to locate subdirectories that contain a satellite DLL for the current culture. If no subdirectory exists, the ResourceManager uses the resource that is embedded in the assembly. US English ("en-US") is the default culture for ActiveReports.

> **Tip**: For more detailed information about how the Framework locates satellite DLLs, please refer to the help system in Visual Studio® or the book *Developing International Software, 2nd edition* by MS Press that contains information on localizing applications using the .NET Framework.

Cultures

For your convenience, here is a list of predefined System.Globalization cultures. (Source: MSDN.) For ActiveReports

localization purposes, use the Culture and Language Name value in the first column.

Culture and Language Name	Culture Identifier	Culture
"" (empty string)	0x007F	Invariant culture
af	0x0036	Afrikaans
af-ZA	0x0436	Afrikaans (South Africa)
sq	0x001C	Albanian
sq-AL	0x041C	Albanian (Albania)
ar	0x0001	Arabic
ar-DZ	0x1401	Arabic (Algeria)
ar-BH	0x3C01	Arabic (Bahrain)
ar-EG	0x0C01	Arabic (Egypt)
ar-IQ	0x0801	Arabic (Iraq)
ar-JO	0x2C01	Arabic (Jordan)
ar-KW	0x3401	Arabic (Kuwait)
ar-LB	0x3001	Arabic (Lebanon)
ar-LY	0x1001	Arabic (Libya)
ar-MA	0x1801	Arabic (Morocco)
ar-OM	0x2001	Arabic (Oman)
ar-QA	0x4001	Arabic (Qatar)
ar-SA	0x0401	Arabic (Saudi Arabia)
ar-SY	0x2801	Arabic (Syria)
ar-TN	0x1C01	Arabic (Tunisia)
ar-AE	0x3801	Arabic (U.A.E.)
ar-YE	0x2401	Arabic (Yemen)
hy	0x002B	Armenian
hy-AM	0x042B	Armenian (Armenia)
az	0x002C	Azeri
az-Cyrl-AZ	0x082C	Azeri (Azerbaijan, Cyrillic)
az-Latn-AZ	0x042C	Azeri (Azerbaijan, Latin)
eu	0x002D	Basque
eu-ES	0x042D	Basque (Basque)
be	0x0023	Belarusian

be-BY	0x0423	Belarusian (Belarus)
bg	0x0002	Bulgarian
bg-BG	0x0402	Bulgarian (Bulgaria)
ca	0x0003	Catalan
ca-ES	0x0403	Catalan (Catalan)
zh-HK	0x0C04	Chinese (Hong Kong SAR, PRC)
zh-MO	0x1404	Chinese (Macao SAR)
zh-CN	0x0804	Chinese (PRC)
zh-Hans	0x0004	Chinese (Simplified)
zh-SG	0x1004	Chinese (Singapore)
zh-TW	0x0404	Chinese (Taiwan)
zh-Hant	0x7C04	Chinese (Traditional)
hr	0x001A	Croatian
hr-HR	0x041A	Croatian (Croatia)
cs	0x0005	Czech
cs-CZ	0x0405	Czech (Czech Republic)
da	0x0006	Danish
da-DK	0x0406	Danish (Denmark)
dv	0x0065	Divehi
dv-MV	0x0465	Divehi (Maldives)
nl	0x0013	Dutch
nl-BE	0x0813	Dutch (Belgium)
nl-NL	0x0413	Dutch (Netherlands)
en	0x0009	English
en-AU	0x0C09	English (Australia)
en-BZ	0x2809	English (Belize)
en-CA	0x1009	English (Canada)
en-029	0x2409	English (Caribbean)
en-IE	0x1809	English (Ireland)
en-JM	0x2009	English (Jamaica)
en-NZ	0x1409	English (New Zealand)
en-PH	0x3409	English (Philippines)

en-ZA	0x1C09	English (South Africa)
en-TT	0x2C09	English (Trinidad and Tobago)
en-GB	0x0809	English (United Kingdom)
en-US	0x0409	English (United States)
en-ZW	0x3009	English (Zimbabwe)
et	0x0025	Estonian
et-EE	0x0425	Estonian (Estonia)
fo	0x0038	Faroese
fo-FO	0x0438	Faroese (Faroe Islands)
fa	0x0029	Farsi
fa-IR	0x0429	Farsi (Iran)
fi	0x000B	Finnish
fi-FI	0x040B	Finnish (Finland)
fr	0x000C	French
fr-BE	0x080C	French (Belgium)
fr-CA	0x0C0C	French (Canada)
fr-FR	0x040C	French (France)
fr-LU	0x140C	French (Luxembourg)
fr-MC	0x180C	French (Monaco)
fr-CH	0x100C	French (Switzerland)
gl	0x0056	Galician
gl-ES	0x0456	Galician (Spain)
ka	0x0037	Georgian
ka-GE	0x0437	Georgian (Georgia)
de	0x0007	German
de-AT	0x0C07	German (Austria)
de-DE	0x0407	German (Germany)
de-LI	0x1407	German (Liechtenstein)
de-LU	0x1007	German (Luxembourg)
de-CH	0x0807	German (Switzerland)
el	0x0008	Greek
el-GR	0x0408	Greek (Greece)

gu	0x0047	Gujarati
gu-IN	0x0447	Gujarati (India)
he	0x000D	Hebrew
he-IL	0x040D	Hebrew (Israel)
hi	0x0039	Hindi
hi-IN	0x0439	Hindi (India)
hu	0x000E	Hungarian
hu-HU	0x040E	Hungarian (Hungary)
is	0x000F	Icelandic
is-IS	0x040F	Icelandic (Iceland)
id	0x0021	Indonesian
id-ID	0x0421	Indonesian (Indonesia)
it	0x0010	Italian
it-IT	0x0410	Italian (Italy)
it-CH	0x0810	Italian (Switzerland)
ja	0x0011	Japanese
ja-JP	0x0411	Japanese (Japan)
kn	0x004B	Kannada
kn-IN	0x044B	Kannada (India)
kk	0x003F	Kazakh
kk-KZ	0x043F	Kazakh (Kazakhstan)
kok	0x0057	Konkani
kok-IN	0x0457	Konkani (India)
ko	0x0012	Korean
ko-KR	0x0412	Korean (Korea)
ky	0x0040	Kyrgyz
ky-KG	0x0440	Kyrgyz (Kyrgyzstan)
lv	0x0026	Latvian
lv-LV	0x0426	Latvian (Latvia)
lt	0x0027	Lithuanian
lt-LT	0x0427	Lithuanian (Lithuania)
mk	0x002F	Macedonian

mk-MK	0x042F	Macedonian (Macedonia, FYROM)
ms	0x003E	Malay
ms-BN	0x083E	Malay (Brunei Darussalam)
ms-MY	0x043E	Malay (Malaysia)
mr	0x004E	Marathi
mr-IN	0x044E	Marathi (India)
mn	0x0050	Mongolian
mn-MN	0x0450	Mongolian (Mongolia)
no	0x0014	Norwegian
nb-NO	0x0414	Norwegian (Bokmål, Norway)
nn-NO	0x0814	Norwegian (Nynorsk, Norway)
pl	0x0015	Polish
pl-PL	0x0415	Polish (Poland)
pt	0x0016	Portuguese
pt-BR	0x0416	Portuguese (Brazil)
pt-PT	0x0816	Portuguese (Portugal)
pa	0x0046	Punjabi
pa-IN	0x0446	Punjabi (India)
ro	0x0018	Romanian
ro-RO	0x0418	Romanian (Romania)
ru	0x0019	Russian
ru-RU	0x0419	Russian (Russia)
sa	0x004F	Sanskrit
sa-IN	0x044F	Sanskrit (India)
sr-Cyrl-CS	0x0C1A	Serbian (Serbia, Cyrillic)
sr-Latn-CS	0x081A	Serbian (Serbia, Latin)
sk	0x001B	Slovak
sk-SK	0x041B	Slovak (Slovakia)
sl	0x0024	Slovenian
sl-SI	0x0424	Slovenian (Slovenia)
es	0x000A	Spanish
es-AR	0x2C0A	Spanish (Argentina)

es-BO	0x400A	Spanish (Bolivia)
es-CL	0x340A	Spanish (Chile)
es-CO	0x240A	Spanish (Colombia)
es-CR	0x140A	Spanish (Costa Rica)
es-DO	0x1C0A	Spanish (Dominican Republic)
es-EC	0x300A	Spanish (Ecuador)
es-SV	0x440A	Spanish (El Salvador)
es-GT	0x100A	Spanish (Guatemala)
es-HN	0x480A	Spanish (Honduras)
es-MX	0x080A	Spanish (Mexico)
es-NI	0x4C0A	Spanish (Nicaragua)
es-PA	0x180A	Spanish (Panama)
es-PY	0x3C0A	Spanish (Paraguay)
es-PE	0x280A	Spanish (Peru)
es-PR	0x500A	Spanish (Puerto Rico)
es-ES	0x0C0A	Spanish (Spain)
es-ES_tradnl	0x040A	Spanish (Spain, Traditional Sort)
es-UY	0x380A	Spanish (Uruguay)
es-VE	0x200A	Spanish (Venezuela)
sw	0x0041	Swahili
sw-KE	0x0441	Swahili (Kenya)
sv	0x001D	Swedish
sv-FI	0x081D	Swedish (Finland)
sv-SE	0x041D	Swedish (Sweden)
syr	0x005A	Syriac
syr-SY	0x045A	Syriac (Syria)
ta	0x0049	Tamil
ta-IN	0x0449	Tamil (India)
tt	0x0044	Tatar
tt-RU	0x0444	Tatar (Russia)
te	0x004A	Telugu
te-IN	0x044A	Telugu (India)

th	0x001E	Thai
th-TH	0x041E	Thai (Thailand)
tr	0x001F	Turkish
tr-TR	0x041F	Turkish (Turkey)
uk	0x0022	Ukrainian
uk-UA	0x0422	Ukrainian (Ukraine)
ur	0x0020	Urdu
ur-PK	0x0420	Urdu (Pakistan)
uz	0x0043	Uzbek
uz-Cyrl-UZ	0x0843	Uzbek (Uzbekistan, Cyrillic)
uz-Latn-UZ	0x0443	Uzbek (Uzbekistan, Latin)
vi	0x002A	Vietnamese
vi-VN	0x042A	Vietnamese (Vietnam)

Section 508 Compliance

Section 508 requires that when Federal agencies develop, procure, maintain, or use electronic and information technology, Federal employees with disabilities have access to and use of information and data that is comparable to the access and use by Federal employees without disabilities, unless an undue burden would be imposed on the agency. Section 508 also requires that individuals with disabilities seeking information or services from a Federal agency have access to and use of information and data that is comparable to that provided to the general public, unless an undue burden would be imposed on the agency.

Summary

Guideline		Applicable	ActiveReports Area
Section 1194.21	Software Applications and Operating Systems	Applicable	End User Designer, Windows Viewer, WPF Viewer, Web controls (HTML Viewer).
Section 1194.22	Web-Based Intranet and Internet Information and Applications	Applicable	Web controls (HTML Viewer).
Section 1194.23	Telecommunications Products	Not applicable	none
Section 1194.24	Video and Multimedia Products	Not applicable	none
Section 1194.25	Self-Contained, Closed Products	Not applicable	none

Section 1194.26	Desktop and Portable Computers	Not applicable	none
Section 1194.31	Functional Performance Criteria	Applicable	End User Designer, Windows Viewer, WPF Viewer, Web controls, HTML Viewer).
Section 1194.41	Information, Documentation and Support	Applicable	Documentation and Support.

Accessibility Summary:

All major features of ActiveReports software are accessible via keyboard navigation.

DISCLAIMER:

GRAPECITY MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS DOCUMENT. The following information reflects the general accessibility features of GrapeCity software components as related to the Section 508 standards. If you find that the information is not accurate, or if you have specific accessibility needs that our products do not meet, please contact us and we will attempt to rectify the problem, although we cannot guarantee that we will be able to do so in every case.

Software Applications and Operating Systems

Section 1194.21

Criteria	Status	Remarks
(a) When software is designed to run on a system that has a keyboard, product functions shall be executable from a keyboard where the function itself or the result of performing a function can be discerned textually.	Supported with exceptions	ActiveReports partially supports keyboard navigation. Users cannot execute report functions completely using the keyboard.
(b) Applications shall not disrupt or disable activated features of other products that are identified as accessibility features, where those features are developed and documented according to industry standards. Applications also shall not disrupt or disable activated features of any operating system that are identified as accessibility features where the application programming interface for those accessibility features has been documented by the manufacturer of the operating system and is available to the product developer.	Supported	The controls do not disrupt or disable industry standard accessibility features of other products.
(c) A well-defined on-screen indication of the current focus shall be provided that moves among interactive interface elements as the input focus changes. The focus shall be programmatically exposed so that Assistive Technology can track focus and focus changes.	Supported with exceptions	Some elements in ActiveReports cannot track focus. Focus is not exposed to screen readers like NVDA.

(d) Sufficient information about a user interface element including the identity, operation and state of the element shall be available to Assistive Technology. When an image represents a program element, the information conveyed by the image must also be available in text.	Supported with exceptions	The images representing program elements do not support alternative text. Partial support for Assistive Technology is provided for some interface elements.
(e) When bitmap images are used to identify controls, status indicators, or other programmatic elements, the meaning assigned to those images shall be consistent throughout an application's performance.	Supported	Any image used to identify a programmatic element has a consistent meaning throughout an application's performance.
(f) Textual information shall be provided through operating system functions for displaying text. The minimum information that shall be made available is text content, text input caret location, and text attributes.	Supported	Textual information such as text content, caret location, and any text attribute (i.e. bold, italic, size, color, etc.) is available for all the viewer elements.
(g) Applications shall not override user selected contrast and color selections and other individual display attributes.	Supported with exceptions	User selected contrast and color settings are not overridden, except for some minor exceptions in End User Designer.
(h) When animation is displayed, the information shall be displayable in at least one non-animated presentation mode at the option of the user.	Supported	Animated image is only used when a report is being loaded in the viewer. The users can provide a static text for the animation using the loadCompleted event of the Viewer.
(i) Color coding shall not be used as the only means of conveying information, indicating an action, prompting a response, or distinguishing a visual element.	Supported	Any interface element that uses color to indicate an action, prompt a response, or distinguish a visual element also provides a textual cue.
(j) When a product permits a user to adjust color and contrast settings, a variety of color selections capable of producing a range of contrast levels shall be provided.	Supported	ActiveReports supports a variety of colors, themes and styles that can be applied to controls in a report.
(k) Software shall not use flashing or blinking text, objects, or other elements having a flash or blink frequency greater than 2 Hz and lower than 55 Hz.	Supported	The controls do not use flashing or blinking text or objects.

(l) When electronic forms are used, the form shall allow people using Assistive Technology to access the information, field elements, and functionality required for completion and submission of the form, including all directions and cues.	Supported	Any form-type dialogs or windows associated with the controls provide Assistive Technology with access to information on all directions, cues, field elements, and functionality required for completion.

Web-based Internet Information and Applications

Section 1194.22

Criteria	Status	Remarks
(a) A text equivalent for every non-text element shall be provided (e.g., via "alt", "longdesc", or in element content).	Supported	Each non-text element has a text equivalent.
(b) Equivalent alternatives for any multimedia presentation shall be synchronized with the presentation.	Not applicable	ActiveReports web controls do not include multimedia.
(c) Web pages shall be designed so that all information conveyed with color is also available without color, for example from context or markup.	Not applicable	ActiveReports web controls do not provide any information using color.
(d) Documents shall be organized so they are readable without requiring an associated style sheet.	Supported with exceptions	Toolbar icons in some web controls are not visible when the associated styles are disabled.
(e) Redundant text links shall be provided for each active region of a server-side image map.	Not applicable	There are no server-side image maps in ActiveReports web controls.
(f) Client-side image maps shall be provided instead of server-side image maps except where the regions cannot be defined with an available geometric shape.	Not applicable	ActiveReports does not use client-side image maps in ActiveReports web controls.
(g) Row and column headers shall be identified for data tables.	Not applicable	By default, there are no data tables associated with the ActiveReports web controls.
(h) Markup shall be used to associate data cells and header cells for data tables that have two or more logical levels of row or column headers.	Not applicable	By default, there are no data tables associated with the ActiveReports web controls.
(i) Frames shall be titled with text that facilitates frame identification and navigation.	Not applicable	By default, there are no frames associated with ActiveReports web controls.
(j) Pages shall be designed to avoid causing the screen to flicker with a frequency greater than 2 Hz and lower than 55 Hz.	Supported	ActiveReports Web controls do not contain any feature that

		causes page flickering.
(k) A text-only page, with equivalent information or functionality, shall be provided to make a web site comply with the provisions of this part, when compliance cannot be accomplished in any other way. The content of the text-only page shall be updated whenever the primary page changes.	Not applicable	Text-only page is not available in ActiveReports WebViewer or Web controls.
(l) When a web page requires that an applet, plug-in or other application be present on the client system to interpret page content, the page must provide a link to a plug-in or applet that complies with §1194.21(a) through (l).	Not applicable	Not applicable
(m) When electronic forms are designed to be completed on-line, the form shall allow people using Assistive Technology to access the information, field elements, and functionality required for completion and submission of the form, including all directions and cues.	Supported with exceptions	The components used in ActiveReports have been configured for maximum compliance. Some form controls may not support Assistive Technology.
(n) A method shall be provided that permits users to skip repetitive navigation links.	Not applicable	ActiveReports does not support repetitive navigation links.
(o) When a timed response is required, the user shall be alerted and given sufficient time to indicate more time is required.	Not applicable	Timed response is not required.

Performance Criteria

Section 1194.31

Criteria	Status	Remarks
(a) At least one mode of operation and information retrieval that does not require user vision shall be provided, or support for assistive technology used by people who are blind or visually impaired shall be provided.	Supported with exceptions	Partial support is provided for assistive technologies like screen readers.
(b) At least one mode of operation and information retrieval that does not require visual acuity greater than 20/70 shall be provided in audio and enlarged print output working together or independently, or support for assistive technology used by people who are visually impaired shall be provided.	Supported	ActiveReports does not require visual acuity greater than 20/70. It is exposed to magnification tools like the Screen Magnifier that can be used for magnifying the screen.
(c) At least one mode of operation and information retrieval that does not require user hearing shall be provided, or support for assistive technology used by people who are deaf or hard of hearing shall be provided.	Not applicable	ActiveReports does not use any sound output. Hearing is not necessary to use the product.
(d) Where audio information is important for the use of a product, at least one mode of operation and information retrieval shall be provided in an enhanced auditory fashion, or support for assistive hearing devices shall be provided.	Not applicable	ActiveReports does not provide functionalities that require audio or video aids.

(e) At least one mode of operation and information retrieval that does not require user speech shall be provided, or support for assistive technology used by people with disabilities shall be provided.	Not applicable	ActiveReports does not provide any functionality that make use of user speech.
(f) At least one mode of operation and information retrieval that does not require fine motor control or simultaneous actions and that is operable with limited reach and strength shall be provided.	Supported	ActiveReports does not provide any functionality that requires fine motor control or simultaneous action.

Information, Documentation and Support

Section 1194.41

Criteria	Status	Remarks
(a) Product support documentation provided to end-users shall be made available in alternate formats upon request, at no additional charge.	Supported	Documentation is available in four formats: HTML(Web), *.chm, *.PDF, and Help3(Visual Studio help content)
(b) End-users shall have access to a description of the accessibility and compatibility features of products in alternate formats or alternate methods upon request, at no additional charge.	Supported	ActiveReports provides information about accessibility and compatibility features in the documentation. The documentation is exposed to screen readers.
(c) Support services for products shall accommodate the communication needs of end-users with disabilities.	Supported	Support services such as telephone, email and forum support are provided to the customers.

How To

Learn to perform common tasks with ActiveReports with quick how-to topics.

Topic	Content
Page Report/RDL Report How To	Learn how to work with both Page report and RDL report, and perform various page report tasks.
Section Report How To	Learn how to work with Section reports and perform various reporting tasks specific to this type of report.
Localize and Deploy	Learn how to localize each of the Windows and Web controls, how to localize reports and controls, and how to deploy Windows and Web applications.

Page Report/RDL Report How To

Learn to perform common tasks in Page Reports and RDL Reports with quick how-to topics.

Report Data

Learn how to connect to different data sources and add a data set in a Page report or a RDL report.

Report Controls
: Learn how to use controls like the TextBox, Map or Image, and data regions like the Tablix or Table on Page Reports/RDL reports.

Manage Data
: Learn how to manage data in both Page report and RDL report.

Page/RDL Report Scenarios
: Learn how to work with both Page report and RDL report, and perform various report tasks.

Interactivity
: Learn how to add parameters, hyperlinks, bookmarks and perform other report operations.

Common Tasks
: Learn how to execute common report tasks.

Report Data

See step-by-step instructions for performing common tasks in Page Reports and RDL Reports.

In this section

Connect to a Data Source
: Learn how to bind reports to various data sources.

Add a Dataset
: Learn how to add a dataset in a report.

Work with Local Shared Data Sources
: Learn how to create and edit a local shared data source, and connect to a local shared data source.

Bind Reports to a Data Source at Run Time
: Learn how to bind a page report to a data source at run time.

Use Dynamically Built JSON Data Source
: Learn how to dynamically build JSON Data Source by using expressions.

Connect to a Data Source

In a Page report or an RDL report, you can connect to a data source at design time through the Report Explorer. Use the following instructions to connect to various data providers supported in ActiveReports.

These steps assume that you have already added a Page Report/RDL Report in a Visual Studio project. See Quick Start for further information.

To connect to SQL, OLEDB, DataSet, ODBC, and Object data sources

1. In the Report Explorer, right-click the Data Sources node and select the **Add Data Source** option or select **Data Source** from the add button.
2. In the **Report Data Source** dialog that appears, select the **General** page and enter the name of the data source. By default, the data source name is set to DataSource1. This name appears as a child node to the Data Sources node in the Report Explorer.
3. Under **Type,** select the type of data source you want to use.
4. Under **Connection**, enter a Connection String. If you select SQL or OleDBas the data source Type, Connection

Properties, Connection String and Advanced Settings pages appear under Connection. If you select DataSet, ODBC, and Object data source Type, Connection Properties and Connection String pages appear. See Report Data Source Dialog for further details.

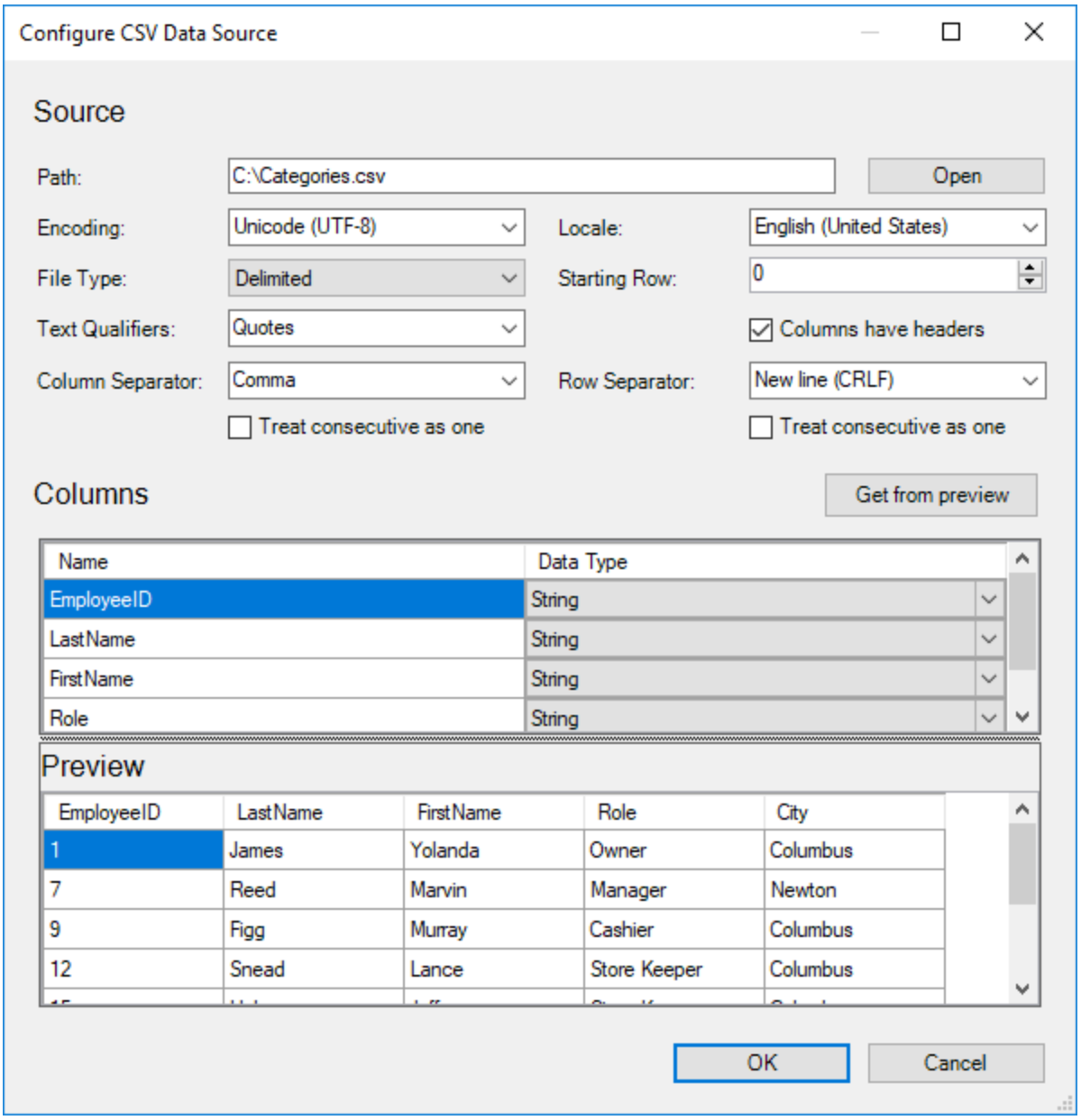
5. Click the **Validate Data Source** icon to confirm the connection string. This icon becomes inactive to indicate success, while an error message indicates an invalid connection string.
6. On the **Credentials** page, you can specify password, credentials, or Windows authentication.
7. Click the **OK** button on the lower right corner to close the dialog. You have successfully connected the report to a data source.

To connect to an XML data source

1. In the Report Explorer, right-click the Data Sources node and select the **Add Data Source** option or select **Data Source** from the add button.
2. In the **Report Data Source** dialog that appears, select the **General** page and enter the name of the data source. By default, the data source name is set to DataSource1. This name appears as a child node to the Data Sources node in the Report Explorer.
3. Under **Type,** select XML Provider.
4. In the **Connection Properties** tab, select the type of XML data from the following options:
    - **External file or URL:** Enter the path of an external XML source such as a local file or the http location of a file.
    - **Embedded:** Enter the path of the XML file to embed in the report. You can also enter the data manually or edit the data in selected XML file.
    - **Expression:** Enter the path expression. User can enter expression in Connection String or in Expression field in Connection Properties.
5. Click the **Connection String** tab. The connection string generated must include xmldoc or xmldata. You can validate the connection string by clicking the **Validate DataSource** icon. For more information on XML connection string, see topic XML Provider.
6. Click the **OK** button on the lower right corner to close the dialog. You have successfully connected the report to a data source.

To connect to a CSV data source

1. In the Report Explorer, right-click the Data Sources node and select the **Add Data Source** option or select **Data Source** from the add button.
2. In the **Report Data Source** dialog that appears, select the **General** page and enter the name of the data source. By default, the data source name is set to DataSource1. This name appears as a child node to the Data Sources node in the Report Explorer.
3. Under **Type,** select CSV Provider.
4. In the **Connection String** tab, click the **Build** icon to open the **Configure CSV Data Source** wizard.
5. Specify the **Path** by clicking the **Open** button and selecting the .csv file available locally, or via URL for centrally located CSV data source. You can also enter a relative path to the csv file here.
6. Set other options in the wizard to generate the connection string. For more information on CSV connection strings, see the CSV Provider topic.
7. To edit the Name and Data Type of columns shown in the Preview area, click the **Get from preview** button. With Fixed data, you can also edit the Width.



8. Click **OK** to save the changes and close the dialog. The **Connection String** tab displays the generated connection string. You can validate the connection string by clicking the **Validate DataSource** icon.
9. Click **OK** on the lower right corner to close the dialog. You have successfully connected the report to a CSV data source. Note that the dataset for the CSV data source is added automatically.

To connect to a JSON data source

1. In the Report Explorer, right-click the Data Sources node and select the **Add Data Source** option or select **Data Source** from the add button.
2. In the **Report Data Source** dialog that appears, select the **General** page and enter the name of the data source. By default, the data source name is set to DataSource1. This name appears as a child node to the Data Sources node in the Report Explorer.
3. Under **Type**, select JSON Provider. See JSON Provider for further details.
4. In the **Schema** tab, specify the JSON schema file corresponding to your JSON data from the following options:

- **External file or URL:** Enter the path or URL of an external JSON schema file or select the file from the drop-down which displays the JSON files available in the same folder as the report.
- **Embedded:** Enter the path of the JSON schema file to embed in the report. You can enter the schema manually or edit the schema in the selected JSON file.
  For generating the JSON schema, use the JSON schema generator available at http://jsonschema.net/.

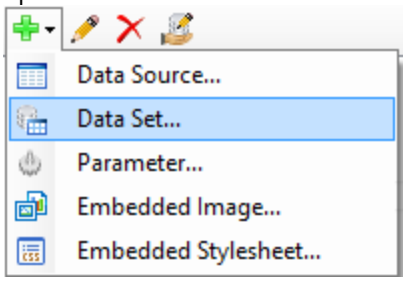
5. In the **Content** tab, specify the JSON data file from the following available options:
   - **External file or URL:** Enter the path or URL of an external JSON data file or select the file from the drop-down which displays the JSON files available in the same folder as the report.
   - **Embedded:** Enter the path of the JSON data file to embed in the report. You can enter the data manually or edit the data in the selected JSON file.
   - **Expression:** Enter an expression to bind to the JSON data.
6. Click the **Connection String** tab. The connection string generated must include jsondoc or jsondata and schemadoc or schemadata, depending on the options selected in Content and Schema tabs. You can validate the connection string by clicking the **Validate DataSource** icon. For more information on JSON connection string, see topic JSON Provider.
7. Click the **OK** button on the lower right corner to close the dialog. You have successfully connected the report to the JSON data source.

Add a Dataset

In a Page report or an RDL report, once you connect your report to a data source, in order to get a list of fields to use in the report, you need to add a dataset. Use the following instructions to add a dataset to the report.

> **Note:** The data set for a CSV data source is automatically created on adding the data source.

These steps assume that you have already added a Page Report/RDL Report template and connected it to a data source. See Quick Start and Connect to a Data Source for further information.

1. In the Report Explorer, right-click the data source node (DataSource1 by default) and select the **Add Data Set** option or select **Data Set...** from the Add button.
   

2. In the DataSet Dialog that appears, select the **General** page and enter the name of the dataset. This name appears as a child node to the data source icon in the Report Explorer.
3. On the **Query** page of this dialog, select a **Command Type** from the dropdown list.
   - **Text -** Allows the user to enter a SQL query or XML path in the **Query** box. See Visual Query Designer for further information on how to create a query using the interactive query designer.
   - **StoredProcedure -** Allows the user to enter the name of the stored procedure in the **Query** box.
   - **TableDirect -** Allows the user to enter the name of the table in the **Query** box.
4. Click the **Validate DataSet** icon at the top right hand corner above the Query box to validate the query.
5. The fields are automatically added to the Fields page in the DataSet dialog. For XML data, manually enter fields on the **Fields** page using valid XPath expressions.

6. You can also set parameters, filters, and data options on the other pages of the dialog.
7. Click the **OK** button to close the dialog. Your data set and queried fields appear as nodes in the Report Explorer.

Note: In case you are using an XML or JSON data source provider, you have to provide an XML path or JSON path using XPath or JSONPath expressions on the Query page, and generate fields on the Fields page of the DataSet dialog. See the following examples for details.

Query and Field settings for XML Data

Connection String

Example of an xmldata connection string.

```
xmldata=<people>
  <person>
    <name>
      <given>John</given>
      <family>Doe</family>
    </name>
  </person>
  <person>
    <name>
      <given>Jane</given>
      <family>Smith</family>
    </name>
  </person>
</people>;
```

XMLPath on Query Page

An XMLPath expression returns a value from an XML data source when evaluated with the query. The XML path is represented by slash (/) and the brackets ([]) represent iteration over collection of elements.

For example: `/people/person/name`

You can also build the XMLPath using **XML DataSet Query Builder**. Click the **Edit with XML Query Designer** icon  to open **XML DataSet Query Builder** dialog and then choose the XPath from the tree nodes.

Fields

Once the query is set, build the Fields collection with two fields that contain the following name and value pairs:
Name: given; **Value:** given
Name: family; **Value:** family

See Add Fields in Reports to understand more about fields.

Query and Field settings for JSON Data

Connection String

Example of a JSON data connection string.

```
jsondoc=C:\Data\customers.json;schemadata={
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "address": {
```

```
        "type": "object",
        "properties": {
          "streetAddress": {
            "type": "string"
          },
          "city": {
            "type": "string"
          }
        },
        "required": [
          "streetAddress",
          "city"
        ]
      },
      "phoneNumber": {
        "type": "array",
        "items": {
          "type": "object",
          "properties": {
            "location": {
              "type": "string"
            },
            "code": {
              "type": "integer"
            }
          },
          "required": [
            "location",
            "code"
          ]
        }
      }
    },
    "required": [
      "address",
      "phoneNumber"
    ]
}
```

JSONPath on Query Page

A JSONPath expression returns a value from a JSON data source when evaluated with the query. The JSON path is usually represented by dot (.) notation, with the root object as '$'. The brackets ([]) represent the array of elements.

For example: `$.Customers[*]`

For more information, please see http://goessner.net/articles/JsonPath/.
You can also build the JSONPath using **JSON Query Builder**, which can be accessed from **Edit with JSON Query Designer** icon  . The JSON Query Builder displays the structure of the JSON data, obtained from the JSON

schema. You can choose the JSONPath from the tree nodes. You can also choose to create multiple datasets - just check-on the **Select multiple nodes** option and select the nodes that you want to add as datasets.

Fields

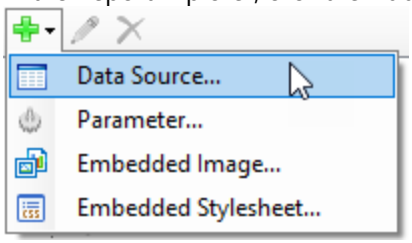
Once the query is set, build the Fields collection with two fields that contain the following name and value pairs:
Name: CompanyName; **Value:** CompanyName
Name: ContactName; **Value:** ContactName

Work with Local Shared Data Sources

You can save a data connection type such as an OleDb connection, Json connection, or an XML connection as a Shared Data Source (RDSX). This topic provides the steps to create a shared data source with an OleDb data connection.

To create a shared data source

1. Create a new Visual Studio project or open an existing one.
2. In the Visual Studio project, add a Page Report or an RDL Report template to create a new report. See Quick Start for further details.
3. In the Report Explorer, click the **Add** icon on the top left and select **Data Source...**



4. In the **Report Data Source** dialog that appears, go to the **General** page and set the following properties to create a connection to an OleDb data source.
   - **Name:** *Any Name*
   - **Type**: Microsoft OleDB Provider

   Under Connection, go to the **Connection Properties** tab and set:

   - **OLE DB Provider**: Microsoft.Jet.OLEDB.4.0
   - **Server or file Name**: *Your .mdb file location*
5. In the **Report Data Source** dialog, click the **OK** button to close the dialog. A data source node appears in the Report Explorer.
6. In the Report Explorer, right-click the data source and select **Share Data Source**.
7. In the **Save Shared Data Source File** dialog that appears, enter the file name and click the **Save** button to save the file in RDSX format. Notice that the data source icon changes to show sharing.

Data Source Icon	Shared Data Source Icon
	

To connect to a shared data source

In ActiveReports, you can connect to most data sources using the steps in the Connect to a Data Source. However, you need to follow the steps below to connect to a Shared Data Source.

1. In the Report Explorer, right-click the Data Sources node and select the **Add Data Source** option or select **Data Source** from the Add button.
2. In the **Report Data Source** dialog that appears, select the **General** page and enter the name of the data source. This name appears as a child node to the Data Sources node in the Report Explorer.
3. Check the **Shared Reference** checkbox on.
4. Under Reference, from the drop-down list, select **From File...**
5. If you have selected the **From File...** option, in the **Shared Data Source File** dialog that appears, go to the folder where your shared data source file is located and select it. A file path appears in the field adjacent to the Browse button.
6. Click the **OK** button on the lower right corner to close the dialog. A shared data source node appears in the Report Explorer.

To edit a shared data source

These steps assume that you have already connected your report to a shared data source.

1. To open the Report Data Source dialog, do one of the following:
   - In the Report Explorer, right-click a shared data source node and in the context menu that appears, select **Edit**.
   - In the Report Explorer toolbar, click the **Edit Shared Data Source** button.
   
2. In the **Report Data Source** dialog that appears, edit the data connection information.

Bind a Page Report to a Data Source at Run Time

ActiveReports allows you to modify a data source at run time. See the following set of sample codes to connect your Page report or RDL report to a data source at run time.

To connect to an OleDB data source

Use the API to set a data source and dataset on a report at run time. These steps assume that you have already added a Page Report template and placed a Viewer control on a Windows Form in your Visual Studio project. See Quick Start and Windows Forms Viewer for further information.

> **Note:** You can use the code example below for the SQL, Odbc, or OleDB data source binding. To do that, modify the Data Provider Type and Connection String according to the data source.

1. From the Visual Studio toolbox, drag and drop a Table data region onto the design surface of the report.
2. In the Table, select the following cells and go to Properties Window to set their Value property.

Cell	Value Property
Left Cell	`=Fields!ProductID.Value`
Middle Cell	`=Fields!InStock.Value`
Right Cell	`=Fields!Price.Value`

3. Go to the Visual Studio Report Menu, and select Save Layout.
4. In the Save As window that appears navigate to your project's folder and save the layout (like RuntimeBinding.rdlx) inside the **bin/debug** folder.
5. Double-click the title bar of the Windows Form to create an event-handling method for the Form_Load event.
6. Add the following code to the handler to connect to a data source, add a dataset and to supply data in the report.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the Form_Load event.

```
'create an empty page report
Dim def As New PageReport
'load the report layout
def.Load(New System.IO.FileInfo(Application.StartupPath + "\RuntimeBinding.rdlx"))
'create and setup the data source
Dim myDataSource As New GrapeCity.ActiveReports.PageReportModel.DataSource
myDataSource.Name = "Example Data Source"
myDataSource.ConnectionProperties.DataProvider = "OLEDB"
myDataSource.ConnectionProperties.ConnectString =
"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=[User
folder]\Samples14\Data\Reels.mdb"
'setup the dataset
Dim myDataSet As New GrapeCity.ActiveReports.PageReportModel.DataSet()
Dim myQuery As New GrapeCity.ActiveReports.PageReportModel.Query()
myDataSet.Name = "Example Data Set"
myQuery.DataSourceName = "Example Data Source"
myQuery.CommandType =
GrapeCity.ActiveReports.PageReportModel.QueryCommandType.TableDirect
myQuery.CommandText =
GrapeCity.ActiveReports.Expressions.ExpressionInfo.FromString("Product")
myDataSet.Query = myQuery
' add fields
Dim _field As New GrapeCity.ActiveReports.PageReportModel.Field("ProductID",
"ProductID", Nothing)
myDataSet.Fields.Add(_field)
_field = New GrapeCity.ActiveReports.PageReportModel.Field("InStock", "InStock",
Nothing)
myDataSet.Fields.Add(_field)
_field = New GrapeCity.ActiveReports.PageReportModel.Field("Price", "Price",
Nothing)
myDataSet.Fields.Add(_field)
'bind the data source and the dataset to the report
def.Report.DataSources.Add(myDataSource)
def.Report.DataSets.Add(myDataSet)
def.Run()
Viewer1.LoadDocument(def.Document)
```

To write the code in C#

C# code. Paste INSIDE the Form_Load event.

```
//create an empty page report
GrapeCity.ActiveReports.PageReport def = new GrapeCity.ActiveReports.PageReport();
//load the report layout
def.Load(new System.IO.FileInfo(Application.StartupPath + "\RuntimeBinding.rdlx"));
//create and setup the data source
GrapeCity.ActiveReports.PageReportModel.DataSource myDataSource = new
GrapeCity.ActiveReports.PageReportModel.DataSource();
myDataSource.Name = "Example Data Source";
myDataSource.ConnectionProperties.DataProvider = "OLEDB";
myDataSource.ConnectionProperties.ConnectString =
"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=[User
folder]\\Samples14\\Data\\Reels.mdb";
//setup the dataset
GrapeCity.ActiveReports.PageReportModel.DataSet myDataSet = new
GrapeCity.ActiveReports.PageReportModel.DataSet();
GrapeCity.ActiveReports.PageReportModel.Query myQuery = new
GrapeCity.ActiveReports.PageReportModel.Query();
myDataSet.Name = "Example Data Set";
myQuery.DataSourceName = "Example Data Source";
myQuery.CommandType =
GrapeCity.ActiveReports.PageReportModel.QueryCommandType.TableDirect;
myQuery.CommandText =
GrapeCity.ActiveReports.Expressions.ExpressionInfo.FromString("Product");
myDataSet.Query = myQuery;
// add fields
GrapeCity.ActiveReports.PageReportModel.Field _field = new
GrapeCity.ActiveReports.PageReportModel.Field("ProductID", "ProductID", null);
myDataSet.Fields.Add(_field);
_field = new GrapeCity.ActiveReports.PageReportModel.Field("InStock", "InStock",
null);
myDataSet.Fields.Add(_field);
_field = new GrapeCity.ActiveReports.PageReportModel.Field("Price", "Price", null);
myDataSet.Fields.Add(_field);
//bind the data source and the dataset to the report
def.Report.DataSources.Add(myDataSource);
def.Report.DataSets.Add(myDataSet);
def.Run();
viewer1.LoadDocument(def.Document);
```

7. Press **F5** to run the Application.

To connect to an unbound Data Source

To connect to unbound data sources at run time, you can use the DataSet provider or the Object provider with the **LocateDataSource** event. The reporting engine raises the **LocateDataSource** event when it needs input on the data to use.

DataSet provider

With the DataSet provider, the ConnectionString and Query settings vary depending on how you connect to data.

To use the LocateDataSource event to bind the report to data, leave the ConnectionString blank.

- If LocateDataSource returns a DataSet, the Query is set to the DataSet table name.
- If LocateDataSource returns a DataTable or DataView, the Query is left blank.

To bind a report to a dataset located in a file, set the ConnectionString to the path of the file, and set the Query to the DataSet table name.

Limitations of the Dataset Provider

- Relationship names that have periods in them are not supported.
- Fields in nested relationships only traverse parent relationships (e.g. FK_Order_Details_Orders.FK_Orders_Customers.CompanyName).

Parent Table Fields

To request a field from a parent table, prefix the field name with the name of the relation(s) that must be traversed to navigate to the appropriate parent table. Separate field names and relations with periods.

For example, consider a main table named OrderDetails which has a parent table named Orders. A relation named Orders_OrderDetails defines the relationship between the two tables. Use a field with the syntax below to access the OrderDate from the parent table:

```
Orders_OrderDetails.OrderDate
```

Use this same technique to traverse multiple levels of table relations. For example, consider that the Orders table used in the prior example has a parent table named Customers and a relation binding the two called Customers_Orders. If the CommandText specifies the main table as OrderDetails, use the following syntax to get the CustomerName field from the parent table:

```
Customers_Orders.Orders_OrderDetails.CustomerName
```

> **Note:** Ambiguity can occur if a field and a relation have the same name. This is not supported.

To use the Dataset Provider

You can use the API to set a dataset on a report at run time.

The Dataset provider returns a data table. All fields in the data table are available. To use the Dataset provider as a report's data source, set up a report definition and run time, and attach the page document to a LocateDataSourceEventHandler.

These steps assume that you have already added a Page Report template and placed a Viewer control on a Windows Form in your Visual Studio project. See Quick Start and Windows Forms Viewer for further information.

1. In the Report Explorer, go to the DataSources node and right-click to select **Add Data Source**.
2. In the Report Data Source dialog that appears, set **Type** to DataSetProvider and close the dialog. A data source node appears in the ReportExplorer.
3. Right-click the data source node and to select **Add Data Set**.
4. In the **DataSet dialog** that appears select the Fields page.
5. On the Fields page, add a fields like `=Fields!ProductID.Value` and `=Fields!InStock.Value.`
6. Click **OK** to close the dialog. Nodes with the field names appear under the dataset name.

7. From the Visual Studio toolbox ActiveReports 14 Page Report tab, drag a Table data region onto the design surface of the report.
8. From the ReportExplorer, add the newly added fields onto cells in the detail row of the Table and save the report.
9. In the Visual Studio Solution Explorer, right-click *YourProjectName* and select **Add > Class**.
10. In the Add New Item window that appears, rename the class to **DataLayer.cs** or **.vb** and click Add.
11. In the Solution Explorer, double-click the DataLayer.cs or .vb to open the code view of the class and paste the following code inside it.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste inside the DataLayer class.

```
Imports GrapeCity.ActiveReports.Expressions.ExpressionObjectModel
Imports System.Globalization
Imports System.Data.OleDb

Friend NotInheritable Class DataLayer
    Private _datasetData As System.Data.DataSet

    Public Sub New()
        LoadDataToDataSet()
    End Sub

    Public ReadOnly Property DataSetData() As System.Data.DataSet
        Get
           Return _datasetData
        End Get
    End Property

    Private Sub LoadDataToDataSet()
        Dim connStr As String = "Provider=Microsoft.Jet.OLEDB.4.0;Persist Security
Info=False;
                Data Source=[User folder]\\Samples14\\Data\\Reels.mdb"
        Dim productSql As String = "SELECT top 100 * FROM Product"

        _datasetData = New DataSet()
        Dim conn As New OleDbConnection(connStr)
        Dim cmd As OleDbCommand = Nothing
        Dim adapter As New OleDbDataAdapter

       cmd = New OleDbCommand(productSql, conn)
       adapter.SelectCommand = cmd
       adapter.Fill(_datasetData, "Products")
    End Sub

End Class
```

To write the code in C#

C# code. Paste inside the DataLayer class.

```
using System;
using System.Data;
using System.Data.OleDb;

internal sealed class DataLayer
{
    private DataSet dataSetData;
        public DataLayer()
        {
                LoadDataToDataSet();
        }

        public DataSet DataSetData
        {
                get { return dataSetData; }
        }

        private void LoadDataToDataSet()
        {
        string connStr = @"Provider=Microsoft.Jet.OLEDB.4.0;Persist Security
Info=False;
                Data Source=[User folder]\\Samples14\\Data\\Reels.mdb";
        string productSql = "SELECT * From Product";

                dataSetData = new DataSet();
        OleDbConnection conn = new OleDbConnection(connStr);
        OleDbCommand cmd = new OleDbCommand(productSql, conn);
        OleDbDataAdapter adapter = new OleDbDataAdapter();
        adapter.SelectCommand = cmd;
        adapter.Fill(dataSetData, "Products");
        }
}
```

Note: The **DataSetDataSource** sample provides context on how to create the DataLayer class, used in the code below. The DataSetDataSource sample can be downloaded from GitHub. See the sample description here.

12. Double-click the title bar of the Windows Form to create an event-handling method for the Form_Load event and add the following code to the handler.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the Form_Load event.

```
LoadReport()
```

Visual Basic.NET code. Paste INSIDE the class declaration of the form.

```
Dim WithEvents runtime As GrapeCity.ActiveReports.Document.PageDocument
```

```
Private Sub LoadReport()
    Dim rptPath As New System.IO.FileInfo("..\..\YourReportName.rdlx")
    'Create a report definition that loads an existing report.
    Dim definition As New GrapeCity.ActiveReports.PageReport(rptPath)
    'Load the report definition into a new page document.
    runtime = New GrapeCity.ActiveReports.Document.PageDocument(definition)
    'Attach the runtime to an event. This line of code creates the event shell
below.
    Viewer1.LoadDocument(runtime)
End Sub

'ActiveReports raises this event when it cannot locate a report's data source in
the usual ways.
Private Sub runtime_LocateDataSource(ByVal sender As Object, ByVal args As
GrapeCity.ActiveReports.LocateDataSourceEventArgs) Handles Runtime.LocateDataSource
    Dim dl = New DataLayer
    args.Data = dl.DataSetData.Tables("Products")
End Sub
```

To write the code in C#

C# code. Paste INSIDE the Form_Load event.

```
LoadReport();
```

C# code. Paste INSIDE the class declaration of the form.

```
private void LoadReport()

{
   System.IO.FileInfo rptPath = new
System.IO.FileInfo("..\\..\\YourReportName.rdlx");
   //Create a report definition that loads an existing report.
   GrapeCity.ActiveReports.PageReport definition = new
GrapeCity.ActiveReports.PageReport(rptPath);
   //Load the report definition into a new page document.
   GrapeCity.ActiveReports.Document.PageDocument runtime = new
GrapeCity.ActiveReports.Document.PageDocument(definition);
   //Attach the runtime to an event. This line of code creates the event shell
below.
   runtime.LocateDataSource += new
GrapeCity.ActiveReports.LocateDataSourceEventHandler(runtime_LocateDataSource);
   viewer1.LoadDocument(runtime);
}

//ActiveReports raises this event when it cannot locate a report's data source in
the usual ways.
private void runtime_LocateDataSource(object sender,
```

```
GrapeCity.ActiveReports.LocateDataSourceEventArgs args)
{

   DataLayer dl = new DataLayer();
   args.Data = dl.DataSetData.Tables["Products"];
}
```

Object Provider

Use the API to bind a report data source to a collection of objects. To bind the Object provider to a report, set up a report definition and a page document, and attach the page document to a LocateDataSourceEventHandler. Create a public class which sets up a property name to which the data field can bind.

The Object provider data source must have a dataset with the Query left blank and fields that correspond to the fields of your Object provider data source. Add these fields manually in the DataSet Dialog under **Fields**.

When using the Object provider, always leave the report's ConnectionString blank because it uses the LocateDataSource event to bind to an Object. Set the Query to one of these values:

To use the Object Provider

These steps assume that you have already added a Page Report template and placed a Viewer control on a Windows Form in your Visual Studio project. See Quick Start and Windows Forms Viewer for further information.

1. In the Report Explorer, go to the DataSources node and right-click to select Add Data Source.
2. In the Report Data Source dialog that appears, set **Type** to ObjectProvider and close the dialog. A data source node appears in the ReportExplorer.
3. Right-click the data source node and in the DataSet dialog that appears select the **Fields** page.
4. In the Fields page, add a field like `=Fields!name.Value` and click ok to close the dialog. A node with the field name appears under the dataset name.
5. From the Visual Studio toolbox ActiveReports 14 Page Report tab, drag a Table data region onto the design surface of the report.
6. From the ReportExplorer, add the newly added field onto a cell in the detail row of the Table.
7. Save the report as **DogReport.rdlx**.
8. In the Solution Explorer, right-click the Form and select **View Code** to open the Code View.
9. In the Code View of the form, paste the following code inside the class declaration.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the class declaration of the form.

```
' Create a class from which to call a property.
Public Class dog
   Private _name As String
   Public Property name() As String
      Get
         Return _name
      End Get
      Set(ByVal value As String)
         _name = Value
      End Set
   End Property
End Class
```

```
' Create an array to contain the data.
Dim dogArray As System.Collections.ArrayList
' Create a method to populate the data array.
Private Sub LoadData()
   dogArray = New System.Collections.ArrayList()
   Dim dog1 As New dog()
   dog1.name = "border collie"
   dogArray.Add(dog1)
   dog1 = New dog()
   dog1.name = "cocker spaniel"
   dogArray.Add(dog1)
   dog1 = New dog()
   dog1.name = "golden retriever"
   dogArray.Add(dog1)
   dog1 = New dog()
   dog1.name = "shar pei"
   dogArray.Add(dog1)
End Sub
```

To write the code in C#

C# code. Paste INSIDE the class declaration of the form.

```
// Create a class from which to call a property.
public class dog
{
   private string _name;
   public string name
  {
      get { return _name; }
      set { _name = value; }
   }
}
// Create an array to contain the data.
System.Collections.ArrayList dogArray;
// Create a method to populate the data array.
private void LoadData()
{
   dogArray = new System.Collections.ArrayList();
   dog dog1 = new dog();
   dog1.name = "border collie";
   dogArray.Add(dog1);
   dog1 = new dog();
   dog1.name = "cocker spaniel";
   dogArray.Add(dog1);
   dog1 = new dog();
   dog1.name = "golden retriever";
   dogArray.Add(dog1);
```

```
    dog1 = new dog();
    dog1.name = "shar pei";
    dogArray.Add(dog1);
}
```

10. Set up the report and add a handler for the LocateDataSource event.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the Form_Load event.

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
    ' Create file info with a path to the report in your project.
    Dim fi As New System.IO.FileInfo("..\\..\\DogReport.rdlx")
    ' Create a report definition using the file info.
    Dim repDef As New GrapeCity.ActiveReports.PageReport(fi)
    ' Create a page document using the report definition.
    Dim runt As New GrapeCity.ActiveReports.Document.PageDocument(repDef)
    ' Create a LocateDataSource event for the runtime.
    AddHandler runt.LocateDataSource, AddressOf runt_LocateDataSource
   ' Display the report in the viewer. The title can be any text.
    Viewer1.LoadDocument(runt)
End Sub
```

To write the code in C#

C# code. Paste INSIDE the Form_Load event.

```
private void Form1_Load(object sender, EventArgs e)
{
    // Create file info with a path to the report in your project.
    System.IO.FileInfo fi = new System.IO.FileInfo("..\\..\\DogReport.rdlx");
    // Create a report definition using the file info.
    GrapeCity.ActiveReports.PageReport repDef = new
GrapeCity.ActiveReports.PageReport(fi);
    // Create a page document using the report definition.
    GrapeCity.ActiveReports.Document.PageDocument runt = new
GrapeCity.ActiveReports.Document.PageDocument(repDef);
    // Create a LocateDataSource event for the runtime.
    runt.LocateDataSource += new
GrapeCity.ActiveReports.LocateDataSourceEventHandler(runt_LocateDataSource);
    // Display the report in the viewer. The title can be any text.
    viewer1.LoadDocument(runt);
}
```

11. Use the LocateDataSource event to load data from the object.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the class declaration of the form.

```
Private Sub runt_LocateDataSource(ByVal sender As Object, ByVal args As
GrapeCity.ActiveReports.LocateDataSourceEventArgs)
   If dogArray Is Nothing Then LoadData()
   args.Data = dogArray
End Sub
```

To write the code in C#

C# code. Paste INSIDE the class declaration of the form.

```
void runt_LocateDataSource(object sender,
GrapeCity.ActiveReports.LocateDataSourceEventArgs args)
{
   if (dogArray == null)
   {
      LoadData();
   }
   args.Data = dogArray;
}
```

12. Press **F5** to run the application.

Use Dynamically Built JSON Data Source

JSON Data Provider supports dynamically built data sources. You can enter a connection string for the JSON data as an expression and pass values using parameters to set up data sources dynamically.

Steps to set up dynamically built data source are as follows:

Note:

- JSON data source used is available at http://jsonplaceholder.typicode.com/comments/.
- JSON schema for the above Json data is generated using http://jsonschema.net.

Create a Page Report

1. Open ActiveReport Report Designer application.
2. From the File menu, select New.
3. In the Create New Report dialog box that appears, select Page Report template and then click OK.

Add a Parameter

4. In the Report Explorer, right-click the Parameters node and select Add Parameters option.
5. In the Report - Parameters dialog that appears, rename the parameter as UserId, and then click OK.

Add a Data Source

6. In the Report Explorer, right-click the Data Sources node and select the **Add Data Source** option or select Data Source from the add button.
7. In the Report Data Source dialog that appears, select the General page and enter the name of the data source. By default, the data source name is set to DataSource1. This name appears as a child node to the Data Sources node in the Report Explorer.
8. Under Type, select JSON Provider.

9. In the **Content** tab, select **Expression**.
10. In the Expression field, enter an expression like the following:

```
="jsondoc=http://jsonplaceholder.typicode.com/comments/" & Parameters!UserId.Value &
";schemadata=
{  ""$schema"": ""http://json-schema.org/draft-04/schema#"",  ""type"": ""object"",
""properties"": {      ""postId"": {        ""type"":    ""integer""      },
      ""id"": {        ""type"":    ""integer""      },
      ""name"": {        ""type"":    ""string""      },
      ""email"": {        ""type"":    ""string""      },
      ""body"": {        ""type"":    ""string""      }    },
    ""required"": [        ""postId"",      ""id"",      ""name"",        ""email"",
""body""    ]  }}"
```

Add a Data Set

11. In the Report Explorer, right-click the data source node (DataSource1 by default) and select the **Add Data Set** option or select **Data Set...** from the Add button.
12. In the DataSet Dialog that appears, select the **General** page and enter the name of the dataset.
13. On the **Query** page of this dialog, select **Command Type** as **Text** and enter **Query** as **$**.
14. On the Fields page, enter the Field name and value pairs as
    Name: postId; Value: postId
    Name: email; Value: email
    Name: name; Value: name
    Name: body; Value: body
15. Click OK.

Create layout for the Report

16. Drop controls from Toolbox onto the report designer surface.
17. Set the Value of each control to a data set field.

Preview the Report

18. Click the Preview tab.
19. Enter the UserId parameter and click **View report**.

Report Controls

Learn to perform common tasks with ActiveReports with quick how-to topics.

In this section

Work with Map
: Learn how to work with layers and data in a Map control.

Work with Images
: Learn how to add an embedded image, data visualizer, web image, or data base image.

Add Table Of Contents
: Learn how to use the TableOfContents control to create an organized hierarchy.

Merge Cells in a Data Region
: Learn how to merge cells in Table and Tablix data regions.

Add Totals and Subtotals in a Data Region
: Learn how to set totals and subtotals in all the data regions with illustrative examples.

Set Fixed Size of a Data Region

Learn how to set the FixedSize property of a Data Region using the resize handler and the Properties window.

Work with Map

Learn using Map data region in Page Reports and RDL Reports.

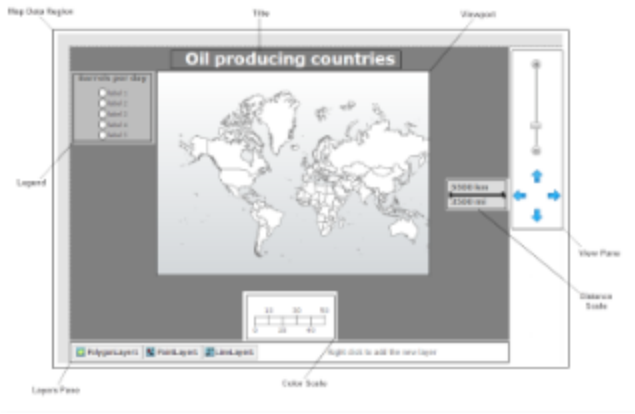
In this section

Create a Map
: Learn how to add and create a Map in a Report.

Add Data
: Learn how to add and work with data in a Map control.

Work with Layers
: Learn how to use different layers to display different data on a Map data region.

Create a Map

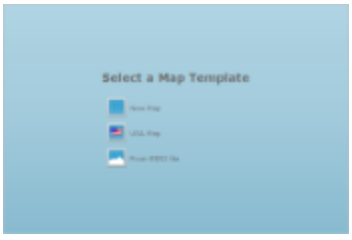


The Map data region shows your business data against a geographical background. This topic illustrates how to create a Map and modify its appearance.

These steps assume that you have added a page layout template to your project and have a data connection in place. See Quick Start and Connect to a Data Source for further information.

To add a map to the report

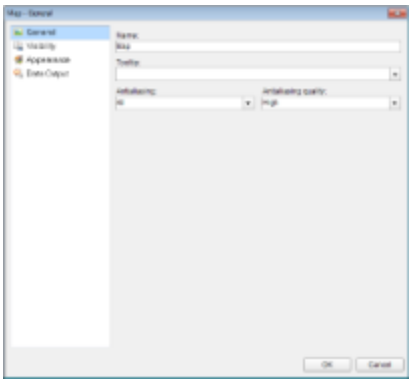
1. From the Visual Studio toolbox, drag a **Map** control onto the design surface.
2. In the **Select a Map Template** wizard that appears, select a map template from the following options:



- **Empty map:** An empty map without any predefined data.
- **USA map:** A map with the predefined polygon layer that contains the embedded spatial data with the USA map.
- **From ESRI file:** Select from your local .shp file, the shapefile spatial data format that complies with the Environmental Systems Research Institute, Inc. (ESRI). An ESRI Shapefile is a collection of files, where a .shp file defines the geographical or geometrical shapes and the .dbf file provides attributes for the shapes in the .shp file. To successfully add spatial data using this option, both files (.shp and .dbf) must be copied to the same folder. ESRI files are available on public domain data sources on the Web, including government and university sites. For more information, go to https://www.census.gov/geographies/mapping-files/time-series/geo/tiger-geodatabase-file.html.

To modify the appearance of the map

1. On the design surface, select the Map control and click the **Property dialog** link in the command section that appears below the properties window.
2. In the **Map** dialog that appears, on the **General** page, enter **"Map"** in the Name textbox. You can also make modifications in properties that control the smoothing mode of all map elements (Antialiasing and Antialiasing quality).

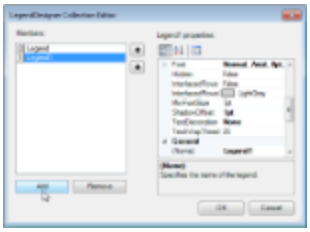


3. On the **Visibility** page of the dialog, you can setup the visibility mode of the map.
4. On the **Appearance** page of the dialog, you can make modifications to the border width, style, color and background color.
5. On the **Data Output** page of the dialog, choose from **Auto**, **Yes** or **No** to decide whether to include this map in the XML output. Also, if you choose to include this map in the XML output, then in **Element Name**, enter a name to be used in the XML output for this map. Choosing **Auto** will include the map in the XML output.
6. Click **OK** to close the dialog.

To add a legend to the map

A legend on a map provides valuable information to users for interpreting the map data visualization rules such as color, size, and marker type differences for map elements on a layer. By default, a single Legend item already exists in the legends collection which can be used by all layers to display items. You can also create additional legends to use them individually with layers that have associated rules to display items in the legend. Use these steps to learn adding and setting a legend on a map:

1. On the design surface, select the Map control.
2. In the Properties window, click the **Legends (Collection)** property and then click the ellipsis (...) button that appears.
3. In the **LegendDesigner Collection Editor** that appears, click **Add** under the Members list of legends. **Legend1** with the default legend settings appears in the Members list.



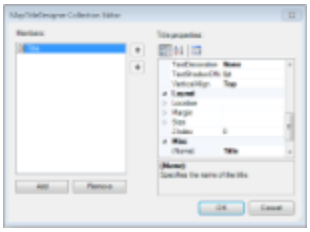
4. With **Legend1** selected in the Members list of legends, you can make modifications to its font, border and background color settings.
5. Click **OK** to close the dialog.

> **Note**: You can associate the newly added legend to a layer by specifying its name in the **Legend Name** field that appears in the Legend tab on a specific rule page of a Layer dialog. See, Use Color Rule, Marker Rule and Size Rule for more information.

To add a title to the map

Map Title describes the theme or subject of the map. The purpose of map title is to tell the viewer of what he is looking at. Use these steps to learn adding a title on a map control.

1. On the design surface, select the Map control.
2. In the Properties window, click the **Titles (Collection)** property and then click the ellipsis (...) button that appears.
3. In the **MapTitleDesigner Collection Editor** that appears, in the Members list of titles, **Title** with the default property settings already exist.



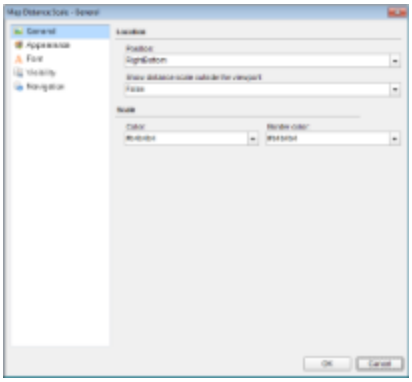
4. In the Properties Window, you can modify the text, font, border and the background color settings of the map title.
5. Click **OK** to close the dialog.

To set the distance scale

A distance scale helps a user to understand the scale of the map. Distance on a map is not the same as the actual real-world distance, so a distance scale shows that a certain distance on the map equals a certain distance in a real-world. In

distance scale, the distance is displayed in both miles and kilometers. The scale range and values are automatically calculated using the viewport boundaries, projection type, and zoom level. Use these steps to learn setting a distance scale on a map:

1. On the design surface, select the Map control.
2. In the Properties window, click the **DistanceScale** property and then click the ellipsis (...) button that appears.
3. In the **Map Distance Scale** dialog that appears, on the **General** page, you can set the location and the color of the distance scale.

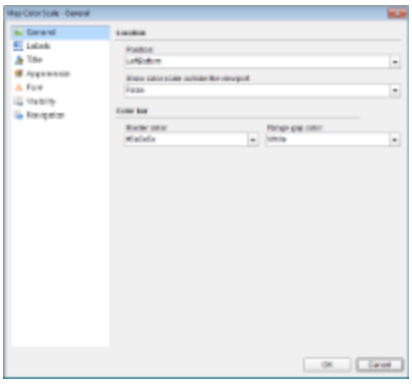


4. On the **Appearance** page of the dialog, you can make modifications to the border width, style, color and background color.
5. On the **Font** page of the dialog, you can make modifications to the font properties of the distance scale.
6. On the **Visibility** page of the dialog, you can setup the visibility mode of the distance scale.
7. On the **Navigation** page of the dialog, you can setup the interactivity features for the distance scale.
8. Click **OK** to close the dialog.

To set the color scale

A color scale helps a user to understand the range of colors that are used for data visualization on a layer. A map has just one color scale and multiple layers can provide data for it. Use these steps to learn setting a color scale on a map.

1. On the design surface, select the Map control.
2. In the Properties window, click the **ColorScale** property and then click the ellipsis (...) button that appears.
3. In the **Map Color Scale** dialog that appears, on the **General** page, you can set the location and the color of the color scale.



4. On the **Labels** page of the dialog, you can make modifications to the properties of the color scale labels.

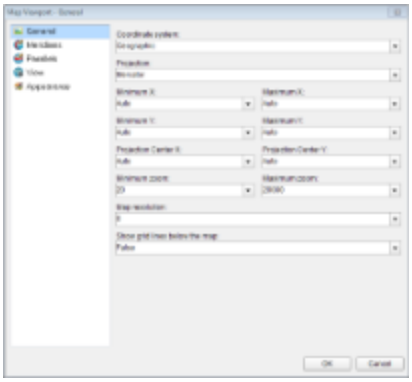
5. On the **Title** page of the dialog, you can make modifications to title text and font properties of the color scale.
6. On to the **Appearance** page, you can make modifications to the border width, style, color and background color.
7. On the **Font** page, you can make modifications to the font properties of the color scale.
8. On the **Visibility** page, you can setup the visibility mode of the color scale.
9. On the **Navigation** page, you can setup the interactivity features for the color scale.
10. Click **OK** to close the dialog.

To modify the appearance of the Viewport

Viewport refers to the area on the map where map data is displayed against a geographical background. It specifies the coordinates, projection system, parallels and meridians, center point, and scale of the map. In other words, it is a map element that actually display geographical data and occupies most area of the map control depending on the location and dock position of other map elements. See Map for more information.

You can modify Viewport properties to make the map look more attractive.

1. On the design surface, select the Map control.
2. Go to the Properties window, click the **Viewport** property and then click the ellipsis (...) button that appears.
3. In the **Map Viewport** dialog that appears, on the **General** page, choose the **Coordinate system**. The map viewport supports the following two coordinate system:



   - **Geographic**: Specifies Earth coordinates by defining longitude and latitude values. If you set the CoordinateSystem property to Geographic, then you must specify the Projection property. A projection is a set of rules on how to locate three-dimensional objects onto a planar surface.
   - **Planar**: Specifies geometric coordinates on a two-dimensional surface by using X and Y values. and in case you set it to Geographic then set the **Projection**.
4. Go to the **Meridians** page, set its visibility and its line and font style and color.
5. Similarly, go to the **Parallels** page, set its visibility and its line and font style and color.
6. On the **View** page of the dialog, choose a Center and Zoom mode. The map viewport supports the following four center and zoom modes:
   - **Custom**: Choose this option to specify custom values for the view center and the zoom level.
   - **Center map to show a map layer**: Choose this option to specify a layer and automatically, center the view on its map data. For example, center the view on LineLayer1.
   - **Center map to show a map element**: Choose this option to center the view on a specific data bound map element. For example, center the view on the map element where the name of the match field is [StateName] and the match value is "Washington".
   - **Center map to show all map elements**: Choose this option to center the view on all map elements in the

layer.

> **Note**: Zoom and View Center level can also be set from the design surface using the zoom slider and arrow keys that appears in the View Pane of the Map control.

7. On the **Appearance** page, set the Border and Background style and color of the viewport.
8. Click **OK** to close the dialog.

Add Data

The Map data region uses the following two types of data:

Spatial Data

Spatial data is a set of coordinates that defines a map element. Each map layer must have spatial data of one of the following types - a polygon, a line, or a point.

Spatial data can be either embedded in a map or can be linked to a map layer. The only difference between the two is that while having the spatial data embedded in a map, there is no separate file to locate or to keep track of when you move the report between projects or machines.

- **Embedded Spatial Data**:

  Embedded spatial data can refer to the following:

  - **Embedded ESRI shapefile or dataset**: When you use the **Embedded** option in Map Layer Data Properties dialog or the **From ESRI file** option in the Map Wizard to import spatial data from an ESRI shapefile, the ESRI shapefile gets embedded in the Map. Similarly, if a report dataset is been used to provide spatial data to a map layer, you always have an option of embedding that spatial data in the map using the **Embed Spatial Data** option that appears in the layers pane. See, Use Layers to learn embedding spatial data to a map.
  - **Custom data**: When you add an empty layer to a map and enter spatial data manually in the LayerDesigner Collection Editor, the data entered gets embedded to the map automatically.

- **External Spatial Data**:

  External spatial data can refer to the following:

  - **Remote or Local ESRI shapefile**: When you use the **Linked** option in Map Layer Data Properties dialog or use the **File** property in the Properties Window to specify the local or remote path/location of an ESRI shapefile, the ESRI shapefile gets linked to the Map layer. However, it remains an external source as the dependent ESRI shapefile needs to moved along with the report between projects or machines.
  - **Report Dataset**: When you use a report dataset to provide spatial data to a map layer and you don't embed the spatial data, it remains an external source. This requires the dependent database file to moved along with the report between projects or machines.

ActiveReports provide numerous ways to add spatial data to the map. You can either use the **Map Wizard** and add data from an ESRI shapefile or use the **Map Layer Data Properties** dialog for using advance options. You can also add spatial data from the Properties Window.

To add spatial data using Map Wizard

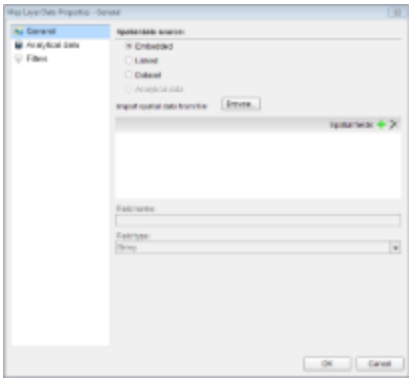
1. From the Visual Studio toolbox, drag and drop a Map control onto the design surface.

2. In the **Select a Map Template** wizard that appears, select **From ESRI file**.
3. In the **Open** dialog that appears, navigate to the folder that contains the .shp and .dbf files, select the .shp file, use the Map Resolution slider to simplify vector data and click **Open**.

This option automatically imports the spatial data stored in the shapefile and adds a related layer to the map control.

To add spatial data using the Map Layer Data Properties dialog

The **Map Layer Data Properties** dialog provide the following advance options to add spatial data. For more information on Map Layer Data Properties dialog, see Map.



- **Embedded** : Use this option where you want to add spatial data from an ESRI file along with an additional option of adding custom spatial data fields if required.

  To add Spatial data from an ESRI file using the Embedded option

  1. In the **Map Layer Data Properties** dialog, on the **General** page, select the **Embedded** option and click the **Browse** button.
  2. In the **Open** dialog that appears, navigate to the folder that contains the .shp and .dbf files.
  3. Select the .shp file, use the Map Resolution slider to simplify vector data and click **Open**. The fields list gets populated with spatial data fields that are added from the shapefile.
  4. In **Dataset**, set the name of the dataset.
  5. In the Fields list, click the Add (+) button to add a new empty spatial data field.
  6. With the newly added spatial data field selected in the list, in **Field name**, enter the name of the field.
  7. In **Field type**, set the field type of newly added spatial data field.
  8. Click **OK** to close the dialog.

  > **Note**: In order to apply the new spatial data field on a layer, you must provide its value for each map layer element like polygons, points or lines in the added map layer Designer Collection Editor dialog.

- **Linked** : Use this option when you just want to add spatial data from an ESRI shapefile without any additional modifications or additions .

  To add Spatial data from an ESRI file using the Linked option

  1. In the **Map Layer Data Properties** dialog, on the **General** page, select the **Linked** option and click the **Browse** button.
  2. In the **Open** dialog that appears, navigate to the folder that contains the .shp and .dbf files.
  3. Select the .shp file and click **Open**.
  4. Click **OK** to close the dialog.

- **Dataset** : Use this option when you have a dataset that stores a spatial data field to provide spatial data to the

Map. You can directly use the data fields from the dataset to display data on a map layer without setting the match fields for analytical data. Therefore, it also provides you an additional option of having different dataset for analytical data and spatial data respectively.

To add Spatial data from a Dataset

1. In the **Map Layer Data Properties** dialog, on the **General** page, select the **Dataset** option.
2. In **Dataset**, set the name of the dataset.
3. In **Field name**, enter the data field name that contains the spatial data.

   > ⚠ **Caution**:
   >
   > - Simply type the name of the dataset field that contains spatial data. For example, enter value as ***StateName***, not as ***=[StateName]***.
   > - You cannot use the MapPoint() function in parameters.

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

> **Note:**
>
> - **MapPoint()** function is supported for Point layer only.
> - Simple (non spatial) data can also be added as Spatial field using MapPoint() expression as `=MapPoint(<Latitude>, <Longitude>)` from Expression Editor.

- **Analytical** : Use this option when you want to use the spatial data field from the same dataset that you may use for adding Analytical data to the layer. For using this option you need to first set the dataset for analytical data and then use the spatial data field from the dataset to provide spatial data to the map layer.

  To add spatial data from Analytical data

  1. Configure Analytical data for the Map control. See the dropdown section below to learn adding Analytical data.

     > **Note**: Once the Analytical data has been set, the Analytical option on the General page of the Map Layer Data Properties dialog becomes active.

  2. In the **Map Layer Data Properties** dialog, on the **General** page, select the **Analytical** option.
  3. In **Field name**, enter the data field name that contains the spatial data.

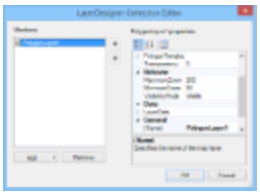
     > ⚠ **Caution**: In **Field name**, enter the data field name as ***=[StateName]***, not as ***StateName***.

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

To add spatial data using Properties Window

These steps assume that you have a Map control containing at least one map data layer placed on the design surface. To learn how to add a layer to the Map control, see Use Layers.

1. With the Map control selected, go to the Properties window, click the **Layers (Collection)** property and then click the ellipsis button that appears.
2. In the **LayerDesigner Collection Editor** that appears, under Members, select the map data layer you want to use and expand the **SpatialData** property.



3. In **Type**, choose the spatial data source of the layer from the following supported options:
    - **Embedded**: Use this option when you want to add custom spatial data to the map data layer. This can be done by first adding the spatial data fields for the embedded spatial data using the MapFieldDefinitionDesigner Collection Editor which can be accessed through the **FieldDefinitions** property. And then later adding spatial data (points, polygons or lines) using the PolygonDesigner Collection Editor, PointsDesigner Collection Editor or LineDesigner Collection Editor depending on the layer in use. These Editor dialogs can be accessed using the **SpatialData** > **Polygons**, **SpatialData** > **Points** or **SpatialData** > **Lines** property.
    - **File**: Use this option when you want to add spatial data from an ESRI file. Use the **Source** property to specify the location of the shapefile.

      > **Note**: The specified location must contain the shape format (.shp) and attribute format (.dbf) files.

    - **DataSet**: Use this option when you have a dataset that stores a spatial data field to provide spatial data to the Map. This option is equivalent to the DataSet option in the **Map Layer Data Properties** dialog. Use the **SpatialData** > **DataSetName** property to specify the name of the dataset and the **SpatialData** > **SpatialField** property to specify the data field that contains spatial data.

      > **Note:**
      >
      > - **MapPoint()** function is supported for Point layer only.
      > - Simple (non spatial) data can also be added as Spatial field using MapPoint() expression as `=MapPoint(<Latitude>, <Longitude>)` from Expression Editor.

      > **Caution:** You cannot use the **MapPoint()** function in parameters.

    - **DataRegion**: Use this option when you have a dataset that stores a spatial data field to provide spatial data to the Map. This option is equivalent to the **Analytical** option in the **Map Layer Data Properties** dialog. Use the **LayerData** > **DataSetName** property to specify the name of the dataset and the **SpatialData** > **VectorData** property to specify the data field or expression that contains spatial data.
4. Click **OK** to close the dialog.

Analytical Data

Analytical data is the data that you want to visualize on the map, for example, tourist attractions in a city or product sales by region. For analytical data, you can associate it with map elements by indicating match fields in the **Match** box of the **Map Layer Data Properties** dialog. You can use one or more fields in the **Match** box of the **Map Layer Data Properties** dialog; for each spatial data field you must indicate a unique analytical data field. This data is optional.

You can get analytical data from the following types of data sources.

- **Dataset field:** A field from a dataset.
- **Spatial data source field:** A field from the spatial data source. For example, you can often find that an ESRI Shapefile contains both spatial and analytical data. Field names from the spatial data source are marked with the # sign in the drop-down list of fields.
- **Embedded data for a map element:** After you embed polygons, lines, or points in a report, you can select the

data fields for map elements and define custom values.

To add Analytical Data to the Map control

Use the following steps to add analytical data to the map. These steps assume that you have added a page layout template to your report and have a data connection in place. See Quick Start and Connect to a Data Source for further information.

1. Add spatial data to the map control. See the dropdown sections above to learn adding spatial data to the map control.
2. In the **Map Layer Data Properties** dialog, go to the **Analytical data** page.
3. Select the dataset that you want to use from the **Dataset** dropdown list and click the **Add** (+) button located next to the **Match** field. This creates an empty match field expression and enables the **Spatial** and **Analytical** field properties.
4. In the **Spatial field** and **Analytical field** options set data fields that contain same data in both Spatial and Analytical databases. This builds the match field expression and relates analytical data to map elements on a map layer.

   > **Note**: It is necessary to set match fields if you want to use a spatial data field from analytical data, or if you want to visualize analytical data on the map layer. Match fields enable the report processor to build a relationship between the analytical data and the spatial data.

5. Click **OK** to close the dialog.

Work with Layers

See step-by-step instructions for using map layers in a Map data region to display data.

In this section

Use Layers
: Learn adding and removing layers, adding bookmarks and hyperlinks to map layer elements and embedding spatial data on a map.

Use a Polygon Layer
: Learn how to use a polygon layer to display data on a map.

Use a Point Layer
: Learn how to use a point layer to display data on a map.

Use a Line Layer
: Learn how to use a line layer to display data on a map.

Use a Tile Layer
: Learn how to use a tile layer to display data on a map.

Use Color Rule, Marker Rule and Size Rule
: Learn how to use Color, Marker and Size rules to modify appearance of the data displayed on a map.

Use Layers

A map is a collection of layers that display data on the map control. See, Map for more information.

This topic illustrates how to add, remove and change order of layers. It also shows how to add interactive navigational feature to map layer elements.

To add a map layer

To add a map layer from the design surface

1. From the Visual Studio toolbox, drag and drop a Map control onto the design surface.
2. In the **Select a Map Template** wizard that appears, select a map template.
3. Click the Map until the map panes appear.
4. Right click inside the area labeled "**Right click to add the new layer**." and select the map layer you want to use.

To add a map layer using the LayerDesigner Collection Editor

1. From the Visual Studio toolbox, drag and drop a Map control onto the design surface.
2. In the **Select a Map Template** wizard that appears, select a map template.
3. With the Map control selected, go to the Properties window, click the **Layers (Collection)** property and then click the ellipsis button that appears.
4. In the **LayerDesigner Collection Editor** that appears, use the **Add** combo-box to view the list of available layers and select the map layer you want to use.

To delete a map layer

To delete a map layer from the design surface

1. On the design surface, click the map until the map panes appear.
2. In the layers pane, right click the layer you want to remove and select **Delete**.

To delete a map layer using the LayerDesigner Collection Editor

1. On the design surface, with the Map control selected, go to the Properties window, click the **Layers (Collection)** property and then click the ellipsis button that appears.
2. In the **LayerDesigner Collection Editor** that appears, under the members list, select the map layer you want to delete and click the **Remove** button.

To change order of layers

Map layers are rendered from left to right in the order that they appear in the map panes. In the image below, the polygon layer is drawn first and the line layer is rendered last. Layers that are rendered later might hide map elements on layers that are rendered earlier. You can change rendering order of layers added to the map control using the **LayerDesigner Collection Editor**. Follow these to steps learn re-ordering the layers on a map.

1. On the design surface, with the Map control selected, go to the Properties window.
2. In the Properties Window, click the **Layers (Collection)** property and then click the ellipsis button that appears.
3. In the **LayerDesigner Collection Editor** that appears, under the members list, select the map layer you want to reorder and use the up or down arrow to change the rendering order of each layer.
4. Click **OK** to close the Collection Editor.

To embed layer spatial data or tiles in a map

When you embed map elements or map tiles in a report, the spatial data is stored in the report definition.

1. Click the map until the map panes appear.
2. In the layers pane, right click the added layer that contains spatial data, select **Embed Spatial Data** and then select **All Spatial Data** or **Currently Visible Data**. In case of Tile layer select **Embed Tiles**.

> **Note**: **All Spatial Data** refers to all the spatial data fields, while **Currently Visible Data** refers to the spatial data field that is set in the **Field** property.

To add Hyperlinks, Bookmarks and drill-through links

Map layer elements like point, polygon and line provides you a functionality to set interactive navigational features like a bookmark link to jump to other areas in the same report, a hyperlink to jump to a Web address, or a drill-through link to jump to another report. Follow these steps to learn adding hyperlinks, bookmarks and drill-through links to a layer element:

1. On the design surface, click the map until the map panes appear.
2. In the layers pane, right click the layer in use and select **Edit**.
3. In the selected layer's dialog that appears, go to the **Navigation** page.
4. On the Navigation page, select from the following actions to perform when a user clicks a data layer element :
    - **None**: The default behavior to indicate that the item has no action.
    - **Jump to report**: For drill-through reporting, select this option and provide the name of a local report, the relative path of a report in another folder, or the full path of a report on another server.
        - **Parameters**: Supply parameters to the targeted report by entering the **Name** of each parameter, the **Value** to send to the targeted report, or whether to **Omit** the parameter. Note that parameter names you supply must exactly match parameters in the target report. You can remove or change the order of parameters using the X and arrow buttons.

      For detailed steps on adding a drill-through link, see Set a Drill-Through Link.

    - **Jump to bookmark**: Select this option and provide a valid **Bookmark ID** to allow the user to jump to another report control with the same Bookmark ID.
      For more information on adding bookmarks, see Add Bookmarks.

    - **Jump to URL**: Select this option and provide a valid URL to create a hyperlink to a Web page.
      For more information on adding hyperlinks, see Add Hyperlinks.

5. Click **OK** to close the dialog.

Use a Polygon Layer

A Polygon layer display outlines of areas or site boundaries that can be linked to Locations and can be used to select records that fall within the boundary for reporting usage.

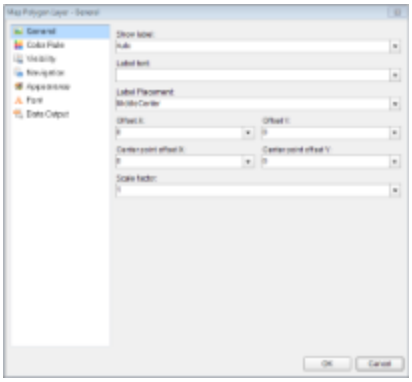
Use the following steps for creating a basic map using the Polygon layer. These steps assume that you have added a page layout template to your project and have a data connection in place. See Quick Start and Connect to a Data Source for further information.

1. From the Visual Studio toolbox, drag and drop a Map control onto the design surface.
2. In the **Select a Map Template** wizard that appears, select the **New Map** template.
3. Click the map until the map panes appear.
4. Right click inside the area labeled "Right click to add the new layer." and select **Add Polygon Layer**. This adds a polygon layer to the map and opens the **Map Layer Data Properties** dialog.

5. In the **Map Layer Data Properties** dialog that appears, on the **General** page, import the spatial data (polygons) from a shape file or use a data field from the analytical data to specify the spatial data source.
6. In case you want to visualize analytical data on the map, go to the **Analytical data** page of the dialog, select your dataset from the **Dataset** dropdown list and click the **Add** (+) button located next to the **Match** field. This creates an empty match field expression and enables the **Spatial** and **Analytical** field properties. See, Add Data for more information.
7. In the **Spatial field** and **Analytical field** options set data fields that contain similar data in both Spatial and Analytical databases. This builds the match field expression and relates analytical data to map elements on a polygon layer.

   > **Note**: It is necessary to set match fields if you want to use a spatial data field from analytical data, or if you want to visualize analytical data on the map layer. Match fields enable the report processor to build a relationship between the analytical data and the spatial data.

8. Go to the **Filters** page and set filters if any.
9. Click **OK** to close the dialog.
10. In the layers pane, right click on **PolygonLayer1** and select **Edit** to open **Map Polygon Layer** dialog.



11. In the **General** page of the dialog, select any data field from the **Label Text** combo box to display as labels inside polygons at run time.
12. Go to the **Color Rule** page of the dialog, to set rules to visualize data using color pallets, color ranges or custom colors for polygons or polygon center points that gets displayed on the map or keep it set to the default "Use Appearance settings". See, Use Color Rule, Marker Rule and Size Rule for more information.
13. Go to the **Visibility** page of the dialog and make sure the layer visibility is set to **Show**. You can also select options to show or hide layer based on any expression or zoom value.
14. In the **Navigation** page of the dialog, you can optionally link the polygon to a URL, bookmark or a report.
15. The **Appearance** page of the dialog either reflects the default appearance of the polygons or the color rule settings if any.
16. In the **Font** page of the dialog, you can optionally set the font family, size, weight, style, set and color for the label text that you had set in the **General** page of the dialog.
17. Go to the **Data Output** page and specify the Data Element Name of the layer to be used while rendering to XML and also specify whether the layer should be included in output while rendering or not.
18. Click **OK** to close the dialog and go to the preview tab to view the map.

Use a Point Layer

A Point layer displays markers for point locations such as a city or an address for a store, restaurant, or school.

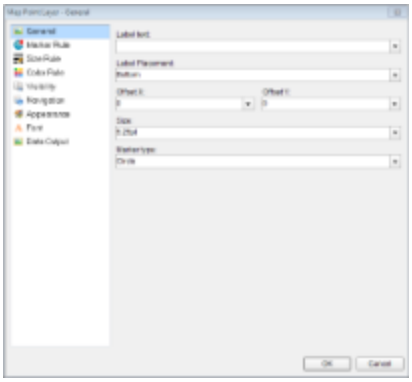
Use the following steps for creating a basic map using the Point layer. These steps assume that you have added a page

layout template to your project and have a data connection in place. See Quick Start and Connect to a Data Source for further information.

1. From the Visual Studio toolbox, drag and drop a Map control onto the design surface.
2. In the **Select a Map Template** wizard that appears, select the **New Map** template.
3. Click the map until the map panes appear.
4. Right click inside the area labeled "Right click to add the new layer." and add either a Tile layer or a Polygon layer. One of these layers that you add serves as the geographic base to plot points on. See, *Use a Polygon Layer* or *Use a Tile Layer* for steps to add these layers on a Map control.
5. Right click again inside the area labeled "Right click to add the new layer." and select **Add Point Layer**. This adds a point layer to the map and opens the **Map Layer Data Properties** dialog.
6. In the **Map Layer Data Properties** dialog that appears, on the **General** page, import the spatial data (points) from a shape file or use a data field from the analytical data to specify the spatial data source.
7. In case you want to visualize analytical data on the map, go to the **Analytical data** page of the dialog, select your dataset from the **Dataset** dropdown list and click the **Add** (+) button located next to the **Match** field. This makes the **Spatial field** and **Analytical field** options active. See, Add Data for more information.
8. In the **Spatial field** and **Analytical field** options set data fields that contain similar data in both Spatial and Analytical databases. Match fields are used to relate the spatial data with the analytical data.

   > **Note**: It is necessary to set match fields if you want to use a data field from the analytical data to set spatial data for the layer, or if you want to visualize analytical data on the map.

9. Go to the **Filters** page and set filters if any.
10. Click **OK** to close the dialog and return to design surface.
11. In the layers pane, right click on **PointLayer1** and select **Edit** to open **Map Point Layer** dialog.



12. In the **General** page of the dialog, select any data field from the **Label Text** dropdown list to display as label for each point that gets displayed on the map at run time. You can also set the label placement, size and marker type.
13. Go to the **Marker Rule** page to set rules to visualize data using markers or keep it set to "Use default marker type".
14. Go to the **Size Rule** page to set rules to visualize data using different marker sizes or keep it set to "Use default marker size". See, Use Color Rule, Marker Rule and Size Rule for more information.
15. Go to the **Color Rule** page to set rules to visualize data using color pallets, color ranges or custom colors for markers that gets displayed on the map or keep it set to "Use Appearance settings".
16. Go to the **Visibility** page of the dialog and make sure the layer visibility is set to **Show**. You can also select options to show or hide layer based on any expression or zoom value.
17. In the **Navigation** page of the dialog, you can optionally link the layer to a URL, bookmark or a report.
18. The **Appearance** page of the dialog either reflects the default appearance of the polygons or the color rule settings if any.
19. In the **Font** page of the dialog, you can optionally set the font family, size, weight, style, set and color for the label

text that you had set in the **General** page of the dialog.

20. Go to the **Data Output** page and specify the Data Element Name of the layer to be used while rendering to XML and also specify whether the layer should be included in output while rendering or not.
21. Click **OK** to close the dialog and go to the preview tab to view the map.

Use a Line Layer

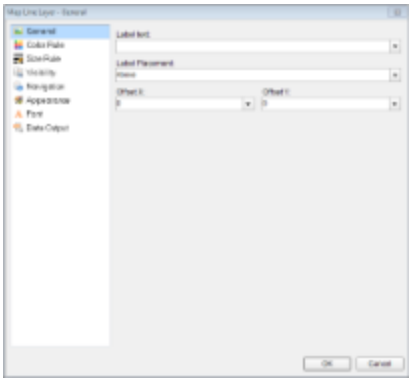
A Line layer displays routes and paths between different locations, for example, transportation route between two stores.

Use the following steps for creating a basic map using the Line layer. These steps assume that you have added a page layout template to your project and have a data connection in place. See Quick Start and Connect to a Data Source for further information.

1. From the Visual Studio toolbox, drag and drop a Map control onto the design surface.
2. In the **Select a Map Template** wizard that appears, select the **New Map** template.
3. Click the map until the map panes appear.
4. Right click inside the area labeled "Right click to add the new layer." and add either a Tile layer or a Polygon layer. One of these layers that you add serves as the geographic base to plot lines on. See, Use a Polygon Layer or Use a Tile Layer for steps to add these layers on a Map control.
5. Right click again inside the area labeled "Right click to add the new layer." and select **Add Line Layer**. This adds a Line layer to the map and opens the **Map Layer Data Properties** dialog.
6. In the **Map Layer Data Properties** dialog that appears, on the **General** page, import the spatial data (lines) from a shape file or use a data field from the analytical data to specify the spatial data source.
7. In case you want to visualize analytical data on the map, go to the **Analytical data** page of the dialog, select your dataset from the **Dataset** dropdown list and click the **Add** (+) button located next to the **Match** field. This makes the **Spatial field** and **Analytical field** options active. See, Add Data for more information.
8. In the **Spatial field** and **Analytical field** options set data fields that contain similar data in both Spatial and Analytical databases. Match fields are used to relate the spatial data with the analytical data.

   > **Note**: It is necessary to set match fields if you want to use a data field from the analytical data to set spatial data for the layer, or if you want to visualize analytical data on the map.

9. Go to the **Filters** page and set filters if any.
10. Click **OK** to close the dialog and return to design surface.
11. In the layers pane, right click on **LineLayer1** and select **Edit** to open **Map Line Layer** dialog.



12. In the **General** page of the dialog, select any data field from the **Label Text** dropdown list to display as label for each line that gets displayed on the map at run time. You can also set the label placement.
13. Go to the **Color Rule** page to set rules to visualize data using color pallets, color ranges or custom colors for lines

that gets displayed on the map or keep it set to "Use Appearance settings". See, Use Color Rule, Marker Rule and Size Rule for more information.

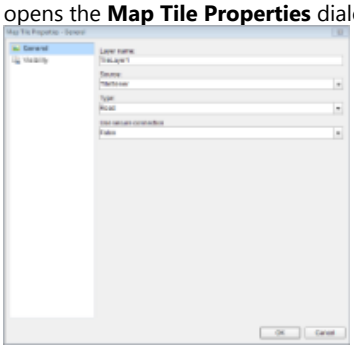
14. Go to the **Size Rule** page to set rules to visualize data using line width or keep it set to the default line width.
15. Go to the **Visibility** page of the dialog and make sure the layer visibility is set to **Show**. You can also select options to show or hide layer based on any expression or zoom value.
16. In the **Navigation** page of the dialog, you can optionally link the layer to a URL, bookmark or a report.
17. Go to the **Appearance** page of the dialog and set the style, width and color of the layer border and the layer background.
18. In the **Font** page of the dialog, you can optionally set the font family, size, weight, style, set and color for the label text that you had set in the **General** page of the dialog.
19. Go to the **Data Output** page and specify the Data Element Name of the layer to be used while rendering to XML and also specify whether the layer should be included in output while rendering or not.
20. Click **OK** to close the dialog and go to the preview tab to view the map.

Use a Tile Layer

A Tile layer displays the virtual earth tile background on the map.

Use the following steps for creating a basic map using the Tile layer. These steps assume that you have added a page layout template to your report and have a data connection in place. See Quick Start and Connect to a Data Source for further information.

1. From the Visual Studio toolbox, drag and drop a Map control onto the design surface.
2. In the **Select a Map Template** wizard that appears, select the **New Map** template.
3. Click the map until the map panes appear.
4. Right click inside the area labeled "Right click to add the new layer." and select **Add Tile Layer**. This adds a tile layer to the map and opens the **Map Tile Properties** dialog.



5. In the **Map Tile Properties** dialog that appears, on the **General** page, set the layer name and choose its Source and Type. The default values in these fields also work fine.
6. In the **Provider** property, choose one from the following supported tile providers:
   - **Bing**: The Microsoft Bing Map server offers static map images. This requires an Application key for authentication. The default key provided by ActiveReports is for demo purpose and can't be used by 3rd party applications. In order to obtain a Bing Map Key, see HowTo - Create a Bing Map Account and HowTo - Get a Bing Map Key.

     Note: After generating the key, add the following script in the Grapecity.ActiveReports.config file to configure embedded Bing tile provider with Application key.

     Script

     Paste inside the <Configuration></Configuration> tags.

     ```
     <MapTileServerConfiguration>
        <Timeout>5</Timeout>
        <AppID>"Your Application Key"</AppID>
     </MapTileServerConfiguration>
     ```

     Caution: The Grapecity.ActiveReports.config file should always be placed in the same folder as the EndUserDesigner.exe file

> for displaying a Bing tile layer on a Map.

- **Google**: The Google Map server creates your map tiles based on URL parameters sent through a standard HTTP request that returns the map tiles as an image. It is necessary to set the API key in order to monitor usage of this tile server with the Google API Console. Please register with Google to get the API key.

> **Note**: After obtaining the key, add the following script in the Grapecity.ActiveReports.config file to configure embedded Google tile provider with ApiKey.
>
> **Script**
>
> Paste inside the <Configuration></Configuration> tags.

```
<!-- Configure embedded Google tile provider with API Key -->
<MapTileProvider Name="Google" DisplayName="Google" >
    <Settings>
      <add key="ApiKey" value="API Key" />
      <add key="Timeout" value="5000" />
    </Settings>
</MapTileProvider>
```

- **CloudMade**: The CloudMade tile server allows you to access it through the HTTP Tile API. The URL structure for this API is straightforward, and is instantly recognizable to anyone familiar with the OpenStreetMap tile numbering convention. To use a CloudMade Tile server you require an API key for authentication which can be obtained by registering at CloudMade.

> **Note**: After generating the key, add the following script in the Grapecity.ActiveReports.config file to configure embedded CloudMade tile provider with ApiKey.
>
> **Script**
>
> Paste inside the <Configuration></Configuration> tags.

```
<!-- Configure embedded CloudMade tile provider with ApiKey -->
<MapTileProvider Name="CloudMade" DisplayName="CloudMade Tiles Provider">
    <Settings>
      <add key="ApiKey" value="API Key" />
    </Settings>
</MapTileProvider>
```

- **MapQuest**: The MapQuest tile server provides the tiles in a format similar to Google. This tile server requires an API Key for authentication which can be obtained by registering at MapQuest.

> **Note**: After generating the key, add the following script in the Grapecity.ActiveReports.config file to configure embedded MapQuest tile provider with ApiKey.
>
> **Script**
>
> Paste inside the <Configuration></Configuration> tags.

```
<!-- Configure embedded MapQuest tile provider with ApiKey -->
<MapTileProvider Name="MapQuest" DisplayName="Map Quest Tiles Provider">
    <Settings>
      <add key="ApiKey" value="API Key" />
      <add key="Timeout" value="3000" />
    </Settings>
</MapTileProvider>
```

- **OpenStreetMap**: The OpenStreetMap server provides the tiles in an index based format. This tile server provides only the roadmap and returns fixed size images (256x256). Before using the OpenStreetMap server, go through the Copyright and License and Tile usage policy pages.

7. Go to the **Visibility** page of the dialog and make sure the layer visibility is set to **Show**. You can also select options to show or hide layer based on any expression or zoom value.

8. Click **OK** to close the dialog and go to the preview tab to view the map.

Note: If you are using proxy server connection to see the map tile images, you need to set credentials for the proxy server in the application config file for authentication. To use your default proxy server credentials, you can do the following:

For Visual Studio Designer (IDE)

1. Find **devenv.exe.config** (the devenv.exe configuration file) in: Program Files (x86)\Microsoft Visual Studio\2017\Professional\Common7\IDE
2. In the configuration file, find the <system.net> block, and add this code:

Paste inside the <system.net></system.net> tags.

```
<defaultProxy useDefaultCredentials="true" />
```

For Visual Studio Application

1. On the menu bar, choose **Project, Add New Item** and then choose the **Application Configuration File** template.
2. In the **Name** text box, enter a name, and then click **Add** button.
3. In the configuration file, add this code:

Paste inside the <configuration></configuration> tags.

```
<system.net>
<defaultProxy useDefaultCredentials="true" />
</system.net>
```

Note: Grapecity.ActiveReports.config file should be kept inside the project **Debug** folder and added to a Visual Studio project for displaying a Tile layer on a Map in any Viewer control.

Add a Custom Tile Provider

You can add and configure a Custom Tile Provider in the Map control using the GrapeCity.ActiveReports.Extensibility.Rendering.Components.Map.**IMapTileProvider** and GrapeCity.ActiveReports.Extensibility.Rendering.Components.Map.**IMapTile** interfaces.

The **IMapTileProvider** interface contains detailed settings that are required to communicate with the tile server, whereas the **IMapTile** interface represents a single tile of a Map's tile layer that fetches the tile image based on the configurations in the **IMapTileProvider** interface.

Adding a custom tile provider also requires making some modifications in the Grapecity.ActiveReports.config file. Follow these steps to learn how to set a custom tile provider:

1. Create a Class Library Project, for example **MyClassLib**, in Visual Studio.
2. Add a new **Class** to the project and name the class, for example, **MyTileProvider**. You may add functions and features to this class for getting the Tile images based on your tile server settings and details. This class serves as the interface between your Map control and your custom tile server. Replace the existing code with the following in the **MyTileProvider** class to implement the **IMapTileProvider** interface.

   To write the code in Visual Basic.NET

VB code. Paste on TOP

```
Imports System
Imports System.Collections.Specialized
Imports GrapeCity.ActiveReports.Extensibility.Rendering.Components.Map
```

VB code. Paste BELOW the Imports statements

```
Namespace MyClassLib
```

```
   Public Class MyTileProvider Implements IMapTileProvider
        ' Tile provider settings, like ApiKey, Language, Style and etc.
        Public Property Settings() As NameValueCollection
        ' Add your code here.
        End Property
        ' Get instance of tile by specifying tile coordinates and details.
        Public Sub GetTile(key As MapTileKey, success As Action(Of IMapTile),
 [error] As Action(Of Exception))
        ' Add your code here.
        End Sub
    End Class

End Namespace
```

To write the code in C#

C# code. Paste on TOP

```
using System;
using System.Collections.Specialized;
using GrapeCity.ActiveReports.Extensibility.Rendering.Components.Map;
```

C# code. Paste BELOW the Using statements

```
namespace MyClassLib
{
    public Class MyTileProvider :IMapTileProvider
    {// Tile provider settings, like ApiKey, Language, Style and etc.
    public NameValueCollection Settings { get; private set;}
    // Get instance of tile by specifying tile coordinates and details.
    public void GetTile(MapTileKey key, Action<IMapTile> success, Action<Exception>
error);
    // Add your code here.
    }
}
```

3. Add a new **Class** to the project and name the class, for example, **MyMapTile**. Replace the existing code with the following in the **MyMapTile** class to implement the **IMapTile** interface.

To write the code in Visual Basic.NET

VB code. Paste on TOP

```
Imports System.IO
Imports GrapeCity.ActiveReports.Extensibility.Rendering.Components.Map
```

VB code. Paste BELOW the Imports statements

```
Namespace MyClassLib
   Public Class MyMapTile Implements IMapTile
       ' Gets the tile identifier
       Public Property Id() As MapTileKey
       ' Add your code here
```

```
        End Property
        ' Gets the tile image stream.
        Public Property Image() As Stream
        ' Add your code here.
        End Property
    End Class
End Namespace
```

To write the code in C#

C# code. Paste on TOP

```
using System.IO;
using GrapeCity.ActiveReports.Extensibility.Rendering.Components.Map;
```

C# code. Paste BELOW the Using statements

```
namespace MyClassLib
    public class MyMapTile : IMapTile
    {   // Gets the tile identifier.
        public MapTileKey Id { get; private set; }
        // Gets the tile image stream.
        public Stream Image { get; private set; }
        // Add your code here.
     }
}
```

4. Add another **Class** to the project and name the class, for example, **WebRequestHelper**. Replace the existing code with the following in the **WebRequestHelper** class to implement the loading of raw website data into the System.IO.MemoryStream class.

To write the code in Visual Basic.NET

VB code. Paste on TOP

```
Imports System.IO
Imports System.Net
```

VB code. Paste BELOW the Imports statements

```
Namespace MyClassLib
    Module StringExtensions
        Public Sub CopyTo(ByVal input As Stream, ByVal output As Stream)
        'Add your code here
        End Sub
        Private Function InlineAssignHelper(Of T)(ByRef target As T, value As T) As T
        'Add your code here
        End Function
    End Module
    Friend NotInheritable Class WebRequestHelper
        Private Sub New()
        End Sub
        ' Load raw data into MemoryStream from specified Url.
        Public Shared Function DownloadData(url As String, timeoutMilliseconds As
```

```
Integer) As Stream
        'Add your code here
        End Function
        'Load raw data into MemoryStream from specified Url.
        Public Shared Sub DownloadDataAsync(url As String, timeoutMilliseconds As
Integer,
 success As Action(Of MemoryStream), [error] As Action(Of Exception))
        'Add your code here
        End Sub
        Private Shared Function InlineAssignHelper(Of T)(ByRef target As T, value As
T) As T
        'Add your code here
        End Function
    End Class
End Namespace
```

To write the code in C#

C# code. Paste on TOP

```
using System.IO;
using System.Net;
```

C# code. Paste BELOW the Using statements

```
namespace MyClassLib
{
    internal static class WebRequestHelper
    {   // Load raw data into MemoryStream from specified Url.
        public static Stream DownloadData(string url, int timeoutMilliseconds)
        {//Add your code here  }
        public static void DownloadDataAsync(string url, int timeoutMilliseconds,
 Action<MemoryStream> success, Action<Exception> error)
        {//Add your code here  }
        public static void CopyTo(this Stream input, Stream output)
        {//Add your code here  }
    }
}
```

5. Save and build your class library project and locate the new .dll file in its **Bin>Debug** folder. This file has the same name as your class library project, with a .dll extension.
6. Create a Basic End User Designer in a new solution following the steps in Creating a Basic End User Designer.
7. Run your Basic End User Designer project to create a EndUserDesigner.exe in your projects **Bin>Debug** folder.
8. Copy the Grapecity.ActiveReports.config file from the *C:\Program Files (x86)\GrapeCity\ActiveReports 14* location and paste it into your End User Designer project's **Bin>Debug** folder.

   > ⚠ **Caution**: Grapecity.ActiveReports.config file should always be placed inside the same folder as the EndUserDesigner.exe file for displaying a tile layer on a Map.

9. Right-click on the Grapecity.ActiveReports.config file and select **Include in this Project** to make changes in the config file.
10. Double-click to open the Grapecity.ActiveReports.config file and paste the following code between the **<Configuration>** and **</Configuration>** tags:

Paste between the <Configuration></Configuration> tags.

```
<!-- Register and configure custom tile provider. -->
<MapTileProvider Name="Custom" DisplayName="Custom Provider" type="YourTileProvider,
AssemblyName,
 Version = x.x.x.x">
    <Settings>
      <add key="ApiKey" value="API Key" />
    </Settings>
</MapTileProvider>
```

> **Note:** Replace ***YourTileProvider*** with fully qualified class name and ***AssemblyName*** with the name of the assembly created after implementing IMapTileProvider and IMapTile interfaces.

11. Add the Class Library project created in step 5 to your Basic End User Designer project.
12. Copy the *YourProjectName.dll* created in step 5 and paste it to the current project's **Bin>Debug** folder together with the EndUserDesigner.exe.
13. Save and Run the project.
14. Create a Report containing a Map control in the **Basic End User Designer**. See Reports with Map for more information.
15. Add a Tile layer to the Map control. Right click the Tile layer and select **Edit** to view the custom tile provider added in the Provider drop-down. See Use a Tile Layer for more information.

Use Color Rule, Marker Rule and Size Rule

You can visualize the data displayed on a map by setting rules to control color, size or marker type for all map elements on a layer. You can set three types of rules depending on the type of layer in use.

Color Rule

Color Rule is set to fill colors for map elements like polygons, markers (points or polygon center points) and lines while using a Polygon, Point or Line layer.
Color Rule provides four options:

1. **Apply Appearance Settings** : Use the default appearance settings that are set in the Appearance page of the map layer dialog.
2. **Visualize data by using color palette** : This option uses an in-built palette that you specify. Based on related analytical data, each map element is assigned a different color from the palette.
3. **Visualize data by using color ranges** : This option, combined with the start, middle, and end colors that you specify on this page and the options that you specify on the Distribution page, divide the related analytical data into ranges. The report then assigns the appropriate color to each map element based on its associated data and the range that it falls into. For example, in a map which uses color to display temperatures on a scale of 0 to 100, low values are blue to represent cold and high values are red to represent hot.
4. **Visualize data by using custom colors** : This option uses the list of custom colors that you specify. Based on related analytical data, each map element is assigned a color from the list.

To set Color Rule for polygons, lines and markers

To visualize polygons, lines or markers using color palette

1. Click the Map until the map panes appear.
2. In the layers pane, right click on the added map layer and select **Edit** to open the selected map layer dialog.

3. In the selected map layer dialog that appears, go to the **Color Rule** page.
4. In the Color Rule page, select the **Visualize data by using color palette** option.
5. In **Data field**, set the name of the field or expression that contains the analytical data that you want to visualize by color.
6. In **Palette**, set the name of the palette to use.
7. Click **OK** to close the dialog.

To visualize polygons, lines or markers using color ranges

1. Click the Map until the map panes appear.
2. In the layers pane, right click on the added map layer and select **Edit** to open the selected map layer dialog.
3. In the selected map layer dialog that appears, go to the **Color Rule** page.
4. In the Color Rule page, select the **Visualize data by using color ranges** option.
5. In **Data field**, set the name of the field or expression that contains the analytical data that you want to visualize by color.
6. In **Start color**, set the color to be used for the color range.
7. In **Middle color**, set the color to be used for the color range.
8. In **End color**, set the color to be used for the color range.
9. Click **OK** to close the dialog.

To visualize polygons, lines or markers using custom colors

1. Click the Map until the map panes appear.
2. In the layers pane, right click on the added map layer and select **Edit** to open the selected map layer dialog.
3. In the selected map layer dialog that appears, go to the **Color Rule** page.
4. In the Color Rule page, select the **Visualize data by using custom colors** option.
5. In **Data field**, set the name of the field that contains the analytical data that you want to visualize by color.
6. Click **Add** to specify each custom color.
7. Click **OK** to close the dialog.

Marker Rule

Marker Rule is set on markers that represent points or polygon center points on a map while using a Point layer.
Marker Rule support two options:

1. **Use a default marker type** : You specify one of the available marker types.
2. **Visualize data using markers** : This option uses a set of markers in an order in which you want them to be used. Marker types include Rectangle, Circle, Diamond, Triangle, Trapezoid, Star, Wedge, Pentagon, PushPin and Image.

To set Marker Rule for points

To visualize points using default marker type

1. Click the Map until the map panes appear.
2. In the layers pane, right click on the added map layer and select **Edit** to open the selected map layer dialog.
3. In the selected map layer dialog that appears, go to the **Marker Rule** page.
4. In **Default**, set a default marker type that will appear in place of each point on a map.
5. Click **OK** to close the dialog.

To visualize points using specific marker types

1. Click the Map until the map panes appear.

2. In the layers pane, right click on the added map layer and select **Edit** to open the selected map layer dialog.
3. In the selected map layer dialog that appears, go to the **Marker Rule** page.
4. In the Marker Rule page, select the **Visualize data using markers** option.
5. In **Data field**, set the name of the field that contains the analytical data that you want to visualize using different marker types.
6. Click **Add** and specify each **Marker type** in an order in which you want them to be used.
7. Click **OK** to close the dialog.

To visualize points using Image as marker type

ActiveReports provides **Image** as one of the many available marker types to use from. You can set this marker type and use any image as a marker on a map layer. Like other marker types, you can either use it as a default marker or use it as one of the member in the markers collection.

Use Image as Default marker type

1. Click the Map until the map panes appear.
2. In the layers pane, right click on the added map layer and select **Edit** to open the selected map layer dialog.
3. In the selected map layer dialog that appears, on the **General** page, set **Marker Type** to **Image**. A new set of properties appears on the page.
4. In **Image Source**, choose the source of image from the provided options:
    - **External**: Select this option and set a path or url of the image file in **Image Value**.
    - **Embedded**: Choose from the list of embedded images added to your report. Once you set this option, the **Image Value** provides you the list of embedded images to choose from.
    - **Database**: Select this option and set the data field containing the image in the **Image Value** property.
5. In **MIME Type**, set the MIME type of the image chosen. In case you are using the Embedded image source the MIME Type gets set automatically as you select the image in the Image Value property.
6. Set the **Transparent Color** and the **Resize Mode**.
7. Click **OK** to close the dialog.

Use Image in markers collection

1. Click the Map until the map panes appear.
2. In the layers pane, right click on the added map layer and select **Edit** to open the selected map layer dialog.
3. In the selected map layer dialog that appears, go to the **Marker Rule** page.
4. In the Marker Rule page, select the **Visualize data using markers** option.
5. In **Data field**, set the name of the field that contains the analytical data that you want to visualize using different marker types.
6. Click **Add** and set **Marker type** to **Image**. A new set of properties for image marker types appears on the page.
7. In **Image Source**, choose the source of image from the provided options:
    - **External**: Select this option and set a path or url of the image file in **Image Value**.
    - **Embedded**: Choose from the list of embedded images added to your report. Once you set this option, the **Image Value** provides you the list of embedded images to choose from.
    - **Database**: Select this option and set the data field containing the image in the **Image Value** property.
8. In **MIME Type**, set the MIME type of the image chosen. In case you are using the Embedded image source the MIME Type gets set automatically as you select the image in the Image Value property.
9. Set the **Transparent Color** and the **Resize Mode**.
10. Click **OK** to close the dialog.

Size Rule

Size Rule is set on markers, polygon center points or line width while using a Polygon, Point or a Line layer.

Size Rule support two options:

1. **Use a default marker size** or **Use a default line width** : You specify the marker size or line width in points.
2. **Visualize data by using size** or **Visualize data by using line width** : In this option, you set the minimum (start) and maximum (end) sizes or width for marker or line, specify the data field to be used for varying the marker size or line width and then specify the distribution options to apply to that data.

To set Size Rule for markers and line width

To visualize marker or line using default marker size or line width

1. Click the Map until the map panes appear.
2. In the layers pane, right click on the added map layer and select **Edit** to open the selected map layer dialog.
3. In the selected map layer dialog that appears, go to the **Size Rule** page.
4. In **Default**, set default size or width for each marker or line that appears on a map.
5. Click **OK** to close the dialog.

To visualize marker or line using specific marker size or line width

1. Click the Map until the map panes appear.
2. In the layers pane, right click on the added map layer and select **Edit** to open the selected map layer dialog.
3. In the selected map layer dialog that appears, go to the **Size Rule** page.
4. In the Size Rule page, select **Visualize data by using size** or **Visualize data by using line width** depending on the layer type in use.
5. In **Data field**, set the name of the field that contains the analytical data that you want to visualize using different marker sizes or line width.
6. Set S**tart size** and **End size** in case of Point Layer or **Minimum line width** and **Maximum line width** in case of Line Layer.
7. Click **OK** to close the dialog.

Distribution Options

The distribution values are used by the rules to differ the map element display values.

To create a distribution of values, you divide your data into ranges by specifying the distribution method, the number of sub-ranges, and the range start and end values.

To set distribution values for rules

1. Click the Map until the map panes appear.
2. In the layers pane, right click on the added map layer and select **Edit** to open the selected map layer dialog.
3. In the selected map layer dialog that appears, go to the rule page(Color Rule, Marker Rule or Size Rule) where you need to specify distribution values.
4. In the rule page of the dialog, select any option to visualize data using the selected rule type and go to the **Distribution** tab.
5. On the Distribution tab, select one of the following distribution types:
    - **EqualInterval** : Create ranges that divide the data into equal range intervals. For the example, the three ranges would be 0-2999, 3000-5999, 6000-8999. Sub-range 1: 1, 10, 200, 500. Sub-range 2: 4777. Sub-range 3: 8999.
    - **EqualDistribution** : Create ranges that divide that data so that each range has an equal number of items. For example, the three ranges would be 0-10, 11-500, 501-8999. Sub-range 1: 1, 10. Sub-range 2: 200, 500. Sub-range 3: 4777, 8999.
    - **Optimal** : Specifies ranges that automatically adjust distribution to create balanced sub-ranges. The number

of Sub-ranges is determined by the algorithm.
    - **Custom** : Specify your own number of ranges to control the distribution of values. For example, you can specify your own custom ranges 0-5000 and 5001-10000.
6. In **Number of Sub-ranges**, type the number of sub-ranges to use.
7. In **Range start**, type a minimum range value. All values less than this number are the same as the range minimum.
8. In **Range end**, type a maximum range value. All values larger than this number are the same as the range maximum.
9. Click **OK** to close the dialog.

Displaying Rule results in Legend

To display rule results in Legend

1. Click the Map until the map panes appear.
2. In the layers pane, right click on the added map layer and select **Edit** to open the selected map layer dialog.
3. In the selected map layer dialog that appears, go to the rule page(Color Rule, Marker Rule or Size Rule) where you need to specify displaying rule results in a legend.
4. In the rule page of the dialog, select any option to visualize data using the selected rule type and go to the **Legend** tab.
5. Select **Show in legend** checkbox and set **Legend name.**
6. In **Legend text**, enter text that specify which data should appear in the legend. Use map keywords and custom formats to help control the format of legend text. For example, #VALUE {C2} specifies a currency format with two decimal places. Following are the supported formats that you can use:

Format	Description	Example
#Value	Displays a numeric value calculated using "**(EndRangeValue - StartRangeValue)/2**" formula.	
#FROMVALUE {C0}	Displays the currency of the total value with no decimal places.	$100
#FROMVALUE {C2}	Displays the currency of the total value to two decimal places.	$40.25
#TOVALUE	Displays the actual numeric value of the data field.	100
#FROMVALUE{N0} - #TOVALUE{N0}	Displays the actual numeric values of the beginning of the range and end of the range.	10 - 500

7. Click **OK** to close the dialog.

Work with Images

The **Image** report control displays an image that you embed in a report, add to a Visual Studio project, store in a database and access through a URL. You can choose an **Image Source** in the Properties window after you place the Image report control on a report.

To embed an image in your report

The benefit of using an embedded image is that there is no separate image file to locate or keep track of when you move the report between projects. The drawback of using embedded images is that when you use large images, it increases the size of your report.

1. In the Report menu, select **Embedded Images**.
2. Click under the **Image** column to reveal an ellipsis button (...) and select an image file from your local files. The **Name** and **MimeType** columns are filled in automatically and the image is stored in the report definition.
3. With the Image report control selected, in the Properties grid, set the **Source** property to **Embedded**.
4. In the **Value** property, select the embedded image from the drop-down list box.

To add a data visualizer image to your report

You can use a data visualizer to display data in small graphs that are easy to comprehend.

1. With the Image report control selected, in the Properties grid, drop down the **Value** property and select **<Data Visualizer...>**.
2. In the Data Visualizers dialog that appears, select the Visualizer Type that you want to use, Icon Set, Range Bar, or Data Bar.
3. Use expressions related to your data to set the other values in the dialog.

To store an image in your Visual Studio project.

You may have an image that you want to use in multiple reports, for example a logo. In such cases, you can store your image as a project image. This not only allows you to quickly locate the correct image for new reports in the project, but also makes it easier when you update your logo, as you will not need to search through every report to replace embedded images. Another benefit is that the images are distributed with your application.

1. From the **Project** menu, select **Add Existing Item** and navigate to the image file that you want to add to the project.
2. With the Image report control selected, in the Properties grid, set the **Source** property to **External**.
3. In the **Value** property, select project image from the drop-down list box.

To use a database image in an Image report control

Product catalogues are probably the most common scenario in which images stored in a database are used in reports. Place the Image report control in a data region to use database images that repeat for every row of data.

> **Note**:
>
> - You cannot use database images in Page Headers and Page Footers because these sections cannot use the value expressions that refer to fields.
> - Microsoft Access database images are generally stored as OLE objects which the Image report control cannot read.

1. With the Image report control selected, in the Properties grid, set the Source property to **Database**.
2. In the **Value** property, select the field containing the image.

To use a Web image

You can also use any image to which you can navigate via a URL. The advantage of using web images is that images stored in this way add nothing to the file size of the project or of the report, but the drawback is that if the web based image is moved, it will no longer show up in your report.

1. With the Image report control selected, in the Properties grid, set the **Source** property to **External**.
2. In the **Value** property, enter the URL for the image.

Add TableOfContents

The Table of Contents control allows you to display the document map, an organized hierarchy of the report bookmark labels and heading levels along with their page numbers, in the body of a report.

To add items to the Document Map using HeadingLevel Property

You can define an hierarchical structure for your report using the **HeadingLevel** property of the TextBox control. For example, you can set the **HeadingLevel** property of the TextBox displaying the report title to **Heading1** and then set the **HeadingLevel** property of the TextBox displaying a group header to **Heading2**, this will club all the Heading2 entries under the Heading1 entry in the Document Map.

These steps assume that you have already set the **HeadingLevel** property of the TextBox controls that your report contains. See, Add Items to the Document Map for more information.

1. In the Report Explorer, right-click the **Report** item and select **Report Properties...**
2. In the Report Properties dialog that appears, go to the **Document Map** page.
3. On the Document Map page, set **Source** to **Headings Only** and optionally select the **Numbering Style** from the dropdown list.

   > **Tip**: You can also set **Source** to **Labels and Headings** to include both labels and headings in the Document Map.

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

To add items to the Document Map using Label Property or Document Map Labels

By setting the **Label** property or **Document Map Label** of the controls in your page report/RDL report, you can show a hierarchical structure based on the parent - child relationship between the controls in the Document Map. For example, set the **Label** property or **Document Map Label** of a List data region and a TextBox control and then place the TextBox control inside the List data region. When you view the Document Map, the TextBox label appears nested inside the List data region label, thereby displaying a hierarchical structure. Similarly, if you set a **Document map label** on multiple Groups of a data region, they appear nested below each other in the document map displaying the same hierarchy in which they were set.

These steps assume that you have already set the **Label** property or **Document Map Label** of the controls that your report contains. See, Add Items to the Document Map for more information.

1. In the Report Explorer, right-click the control and select **Report Properties...**
2. In the Report Properties dialog that appears, go to the **Document Map** page.
3. In the Document Map page, set **Source** to **Labels Only** and optionally select the **Numbering Style** from the dropdown list.

   > **Tip**: You can also set **Source** to **Labels and Headings** to include both labels and headings in the Document Map.

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

To add items to the Document Map

1. On the design surface, select a control you want to add to the Document map.
2. In the command section of the Properties Window, click the Property dialog. This is a command to open the control's dialog. See Properties Window for more information on how to access commands.
3. In the dialog that appears, go to the **Navigation** page and under the **Document map label**, enter a text or an expression representing the control in the Document map.
4. Click **OK** to close the dialog.

To add and configure TableOfContents

Follow these steps to setup the TableOfContents control in the report layout displaying the document map set in the procedures above.

To add a TableOfContents control in Page Report

1. In the Report designer, click the **New** tab to add a new page to the report layout.
2. From the toolbox, drag and drop the **TableOfContents** control onto **Page 1** while use **Page 2** to create layout for displaying the main content of the report. Using the same page that contains the TableOfContents may disrupt the flow of data displayed in the report.
3. On the design surface, select the TableOfContents control and go to the Properties window to set its **FixedHeight** property. The use of the FixedHeight property is similar to the use of the FixedSize property that is available with other report controls.

   > **Note**: You can also place the Overflow Place Holder control and link it with the TableOfContents control using its **OverflowName** property to display data that does not fit inside the fixed size of the TableOfContents control.

4. With the TableOfContents control selected, click the **Levels (Collection)** property and then click the ellipsis button that appears.
5. In the **LevelDesigner Collection Editor** that appears, consider the hierarchy of entries that appear in the Document Map and add as many levels required using the **Add** button. This allows you to customize entries at different nested levels. Using just a single Level will list down all the TableOfContents entries at the same level. You can also set various numbering styles for all levels or an individual level by setting the **Numbering Style** for document map levels using the Report dialog or using the **DocumentMap** property that gets directly applied to the TableOfContents control. For more information see Add Items to the Document Map.
6. Select each level and set its related properties available in the LevelDesigner Collection Editor. These properties could be general properties like DisplayPageNumber or DisplayFillCharacter, or they could be related to the level's appearance like BackgroundColor, Font, Padding etc. These properties directly affect all the entries that appear in the selected level, thereby allowing you to customize them. For information on the important properties of the TableOfContents control, see Table of Contents.

7. Click **OK** to close the dialog and return to the design surface.
8. Go to the Preview Tab to view the Table of Contents appear in the report output.
9. Click any TableOfContents entry and navigate to the targeted report control in the report.

To add a TableOfContents control in Report Definition Language (RDL)

1. From the toolbox, drag and drop the **TableOfContents** control onto the report design surface, preferably at the start or end of the report layout to justify the significance of the control.

To configure TableOfContents Appearance

1. With the TableOfContents control selected, click the **Levels (Collection)** property and then click the ellipsis button that appears.
2. In the **LevelDesigner Collection Editor** that appears, consider the hierarchy of entries that appear in the Document Map and add as many levels required using the **Add** button. This allows you to customize entries at different nested levels. Using just a single Level will list down all the TableOfContents entries at the same level.
3. Select each level and set its related properties available in the LevelDesigner Collection Editor. These properties could be general properties like DisplayPageNumber or DisplayFillCharacter, or they could be related to the level's appearance like BackgroundColor, Font, Padding etc. These properties directly affect all the entries that appear in the selected level, thereby allowing you to customize them. For information on the important properties of the TableOfContents control, see Table of Contents.

   > **Note**: You can also set various numbering styles for all levels or individual level by setting the **Numbering Style** for document map levels using the Report dialog or using the **DocumentMap** property that gets directly applied to the TableOfContents control. For more information see Add Items to the Document Map.

4. Click **OK** to close the dialog and return to the design surface.
5. Go to the Preview Tab to view the Table of Contents appear in the report output.
6. Click any TableOfContents entry and navigate to that targeted report control in the report.

To apply styles to the TableOfContents control

In the TableOfContents control, styles can be applied using the **StyleName** property.

1. Create a new style sheet and add styles that you want to apply to the TableOfContents control. For more information on creating style sheets and style types, see Working with Styles and Style Elements.
2. Apply the style sheet to the report. For details on how to apply style sheets to reports at design time, see Working with Styles.
3. From the toolbox, drag and drop the **TableOfContents** control onto the report design surface, preferably at the start or end of the report layout to justify the significance of the control.
4. On the design surface, select the TableOfContents control.
5. In the Properties Window, from the **StyleName** property drop-down, select a style to apply to the TableOfContents control.

To apply styles to the TableOfContents levels

In the TableOfContents control, styles can be applied to each TableOfContents level using the **StyleName** property available in the **LevelDesigner Collection Editor** dialog.

1. Create a new style sheet and add styles that you want to apply for the TableOfContents level. For more information on creating style sheets and and the type of styles available for TableOfContents level, see Working with Styles and

Style Elements.
2. From the toolbox, drag and drop the **TableOfContents** control onto the report design surface, preferably at the start or end of the report layout to justify the significance of the control.
3. With the TableOfContents control selected, click the **Levels (Collection)** property from the Properties window and then click the ellipsis button that appears.
4. In the **LevelDesigner Collection Edior** dialog that appears, select a TableOfContents level on which to want to apply the style.
5. From the list of properties on the right, drop-down the **StyleName** property to select a style to apply.

Merge Cells in a Data Region

Merging data in Table or Tablix data regions is a common scenario. **ActiveReports 14** provides **AutoMergeMode** property that consists of Always, Never, and Restricted values. The detail cells with same data values and with AutoMergeMode property set to:

- **Always** - are merged.
- **Never** - are not merged.
- **Restricted** - are merged only if the corresponding cells in previous columns are similarly merged. If for example, cells in Column 2 (with same data values) are set 'Restricted' and the corresponding cells (with same data values) in previous column, that is Column 1, are set 'Never', then cells in Column 2 are not merged.

The reports using AutoMerge in older versions can be opened correctly in **ActiveReports 14**, and new reports with AutoMergeMode are correctly opened in previous ActiveReports versions.

The following steps take you through how to add automatic merge to the cells in Table and Tablix data regions.

These steps assume that you have already added a Page Report/RDL Report template to your project, connected it to a data source and added a dataset. See Quick Start, Connect to a Data Source and Add a Dataset for more information.

> **Note**: This topic uses the Orders table in the NWind database. The NWIND.mdb file can be downloaded from GitHub: ..\Samples14\Data\NWIND.mdb.

Order ID	Ship Name	Employee ID
10273	QUICK-Stop	3
10274	Vins et alcools Chevalier	6
10275	Magazzini Alimentari Riuniti	1
10276	Tortuga Restaurante	8
10277	Morgenstern Gesundkost	2
10278	Berglunds snabbköp	8
10279	Lehmanns Marktstand	8
10280	Berglunds snabbköp	2
10281	Romero y tomillo	4
10282		
10283	LILA-Supermercado	3
10284	Lehmanns Marktstand	4
10285	QUICK-Stop	1
10286		8

To apply merge in Table cells

1. From the toolbox, drag a Table data region onto the report design surface.
2. Select the Table and set the **BorderStyle** property to Solid.
3. In the Table, select the following TextBoxes and from the Fields Selection Adorner, set their Values as follows.

TextBox	Value
TextBox4	`OrderID`
TextBox5	`ShipName`
TextBox6	`EmployeeID`

4. Select TextBox6 and set **AutoMergeMode** property to Restricted. This merges employee ids depending on whether the corresponding ship names (cells in previous column) are merged.
5. Select TextBox5 and set AutoMergeMode property to Always. This merges the cells with similar ship names.
6. Set the **BorderStyle** property of Header row and Detail row to Solid to view the merged cells clearly.

To merge cells in Tablix (outside row group)

1. From the toolbox, drag a Tablix data region onto the report design surface.
2. Select the Tablix and set the **BorderStyle** property to Solid.
3. Right-click Textbox4, select **Insert Column**, and then select **Outside Group - Right**.

4. In the Tablix, select the following TextBoxes and set their Values as follows.

TextBox	Value
TextBox3	`OrderID`
TextBox4	`ShipName`
TextBox6	`EmployeeID`

5. Select TextBox6 and set **AutoMergeMode** property to Restricted. This merges employee ids depending on whether the corresponding ship names (cells in previous column) are merged.
6. Select TextBox4 and set AutoMergeMode property to Always. This merges the cells with similar ship names.
7. Select the Row Group area and set the **BorderStyle** property to Solid to view the merged cells clearly.

Add Totals and Subtotals in a Data Region

You can add subtotals and grand totals in a data region to add meaning to the data it displays.

Use the steps below to learn how to set subtotals and totals in each data region. These steps assume that you have already added a Page Report/RDL Report template and connected it to a data source. See Quick Start, Connect to a Data Source and Add a Dataset for more information.

> **Note:** This topic uses examples from the tables in the Reels database. The Reels.mdb file can be downloaded from GitHub: ..\Samples14\Data\Reels.mdb.

To add Totals and Subtotals in a Table

To add a subtotals to a table group

1. From the Visual Studio toolbox, drag and drop a Table data region onto the design surface.
2. From the Report Explorer, drag a numeric field (like *InStock*) onto the detail row of the Table. This is the field for which you want to display subtotals.
3. Follow the steps below to add groups to the Table data region.
    - Click inside the Table to reveal the row and column handles along the left and top edges.
    - Right-click in the row handles along the left edge of the table and select **Insert Group**.
    - In the **Groups** dialog that appears, set a **Group on** expression (like *StorePrice*) on which you want to group the data.
    - Click the **OK** button to close the dialog and add the group. A new pair of rows for group header and footer appear on the Table.
4. From the Report Explorer, drag and drop the numeric field (like *InStock*) you added to the detail row in step 2, onto the **GroupFooter** row.
5. Double click the textbox containing the field you just dropped onto the **GroupFooter** row and add a Sum function to its expression to calculate the total [for example, `=Sum(Fields!InStock.Value)`].
6. Go to the Preview Tab to see the subtotals appearing below each group of data in the Table.

To add a grand total to a table

1. Drag the numeric field (like *InStock*) you used to set subtotals on in the procedure above onto the Table Footer row.
2. Double click the textbox containing the field you just dropped onto the Table Footer row and add a Sum function to its expression to calculate the total [for example, `=Sum(Fields!InStock.Value)`].
3. Go to the Preview Tab and notice that at the end of the table, the Textbox from the Table Footer row supplies the grand total.

To add Totals and Subtotals in a List

To add a subtotals to a list

1. From the Visual Studio toolbox, drag and drop a List data region onto the design surface.
2. From the Report Explorer, drag and drop a numeric field (like *Price*) onto the List data region.
3. Double click the textbox containing the field you just dropped and add a Sum function to its expression to calculate the total [for example, ***=Sum([Price] * [Quantity])***].

> **Note**: If you preview the report at this point, you will notice that the field renders the grand total for the dataset after each sales amount.

4. Follow the steps below to add detail grouping to the List data region.
    - Right-click the list and select **Properties**.
    - In the **Detail Grouping** page, select a **Group on** expression (like *AccountNumber)* on which you want to group the data.
    - Click the **OK** button to close the dialog and apply grouping.
5. Go to the Preview Tab to view the report, you can see a subtotal on price for each account number.

To add a grand total to a list

1. Drag the numeric field that shows subtotals in your list (like *Price*) just below the List data region.
2. Double click the textbox containing the field you just dropped and add a Sum function to its expression to calculate the grand total [for example, ***=Sum([Price] * [Quantity], "List1")***].
3. Go to the Preview Tab and notice that below the List there is a Textbox that supplies the grand total.

To add Totals and Subtotals in a BandedList

To add a subtotal to a banded list

1. From the Visual Studio toolbox, drag and drop a Banded List data region onto the design surface.
2. From the Report Explorer, drag a numeric field (Like *InStock*) onto the detail band of the banded list.
3. Follow the steps below to add groups to the BandedList data region.
    - Right-click the BandedList and select **Insert Group**.
    - In the **Groups** dialog that appears, select a **Group on** expression (like *StorePrice*) on which you want to group the data.
    - Click the **OK** button to close the dialog and add a group. A new pair of group bands appear on the data region.
4. From the Report Explorer, the numeric field (like *InStock*) you added to the detail band in step 2, onto the **GroupFooter** band.
5. Double click the textbox containing the field you just dropped and add a Sum function to its expression to calculate the subtotal[for example, `=Sum(Fields!InStock.Value)`].
6. Go to the Preview Tab to view the report to see the subtotals appearing below each group of data in the BandedList.

To add a grand total to a banded list

1. Drag the numeric field (like InStock) you used to set subtotals on in the procedure above onto the BandedListFooter band.
2. Double click the textbox containing the field you just dropped onto the BandedListFooter band and add a Sum function to its expression to calculate the total [for example, **=Sum(Fields!InStock.Value)**].
3. Go to Preview Tab and notice that at the end of the BandedList, the Textbox from the BandedListFooter band supplies the grand total.

To add Totals and Subtotals in a Tablix

Note: In this example, we are using the **StoreSalesbyYear** table from the Reels database.

To add a subtotal to a Tablix group

1. From the Visual Studio toolbox, drag and drop a Tablix data region onto the design surface.
2. Drag and drop the **StoreName** field from the Report Explorer to the row group area (bottom left corner) of the Tablix data region. This is the row header, and dragging a field into it automatically adds a row group.
3. Drag and drop the **SaleYear** field from the Report Explorer to the column group area (top right corner) of the Tablix data region. This is the column header, and dragging a field into it automatically adds a column group. Modify the **SaleDate** field expression to `=[SaleDate].Year` to provide yearly data.
4. Drag and drop the **TotalSales** field from the Report Explorer to the body area (bottom right corner) of the Tablix data region. This will automatically set expression of the cell to ***=Sum(Fields!TotalSales.Value)***.
5. Right-click the column group area, select **Add Total**, and then click **After.** A new column appears to the right with the text **Total**. This displays the subtotals for each row group.
6. Right-click the row group area, select **Add Total**, and then click **After.** A new row appears at the bottom with the text **Total**. This displays the subtotals for each column group.

Store Sales	=[SaleYear]	Total Store Sales for Year 2004-2005
=[StoreName]	=Sum([TotalSales])	=Sum([TotalSales])
Total Yearly Sales	=Sum([TotalSales])	=Sum([TotalSales])

7. Go to the Preview tab to view the subtotals for each year.

To add a grand total to a Tablix

1. The row and columns subtotals set in the previous procedure, intersect at the rightmost cell of the Tablix that contains the grand total of the combined sales amount for years 2004 and 2005 in all the stores.
2. Go to the Preview tab to view the result.

Set Fixed Size of a Data Region

FixedSize property of a Page Report is used to set the exact height or width (or both) that the data region will occupy at run time. Using this property, you can control the number of records that are displayed on each page.

In case there are more records to display than what **FixedSize** can accommodate, the remaining records get displayed on the next page. However, in case there is an OverflowPlaceHolder control bound to a data region on the same page, the remaining records are displayed at the location where the OverflowPlaceHolder control is placed. When you link a data region to an OverflowPlaceHolder, this control gets its **Size** property value from the **Size** of the data region it is linked with. However, it is also possible to change the Size (Height and Width) property values of OverflowPlaceHolder control based on the requirement of your Page Report.

> ⚠ **Caution:** The Size of an OverflowPlaceHolder cannot be changed to a value which is less than the Size of a data region it is linked with.

The significance of the **FixedHeight** and **FixedWidth** property depends on the data region that is being used. For example, in case of Table data region, only the FixedHeight property holds importance as the Table control always grows vertically to accommodate excessive data, while in case of Tablix data region both FixedHeight and FixedWidth properties are important as the control expands both vertically and horizontally to fit excessive data.

The table below show the data regions that contains the FixedSize property and their support for FixedHeight and FixedWidth properties:

Data Region	Supported Property

Table	FixedHeight
Tablix	FixedHeight and FixedWidth both
List	FixedHeight
BandedList	FixedHeight

To set the FixedSize Property

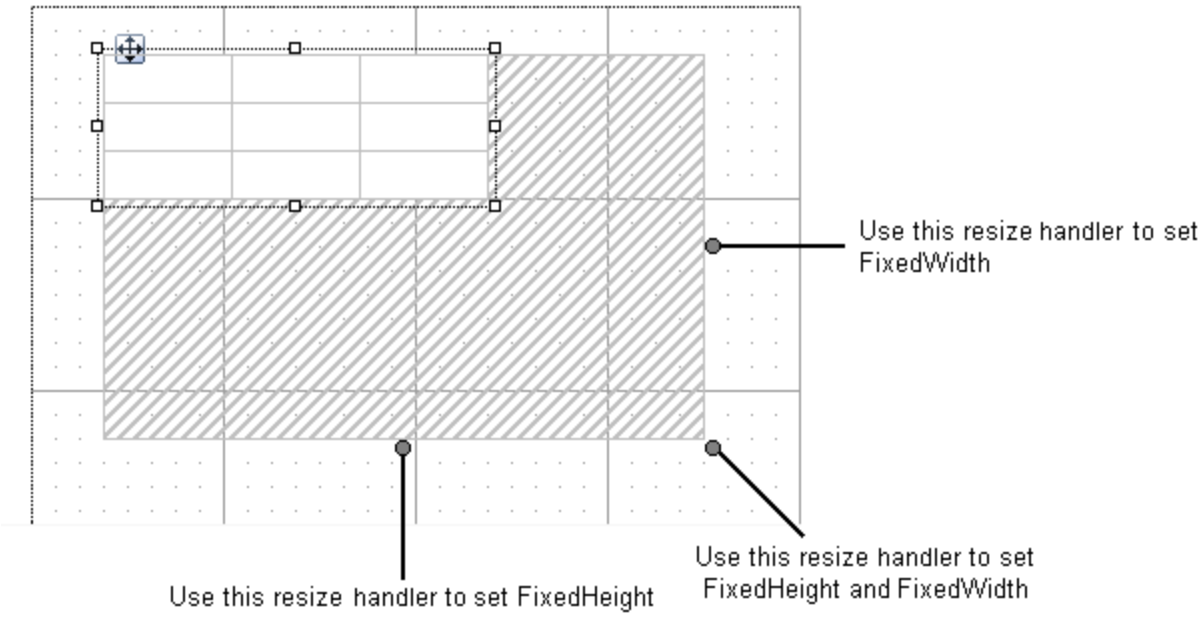
You can use the Properties Window to set the **FixedHeight** and **FixedWidth** properties of a data region manually, or use the resize handlers that appears around the data region to set the FixedSize.

Follow these steps to set FixedSize through the Properties window:

1. From the Visual Studio toolbox, drag and drop a data region like List or a Table control onto the design surface.
2. On the design surface, click the data region to select it and go to the Properties Window.
3. In the Properties Window, locate and expand the **FixedSize** property node to assign values for the **Height** and **Width**.

Follow these steps to set FixedSize using the resize handlers:

1. From the Visual Studio toolbox, drag and drop a data region like List or a Table control onto the design surface. Notice the resize handlers that appear around the data region.
2. Use mouse to drag the resize handlers to set the **FixedSize** area of a data region.



Manage Data

See step-by-step instructions for performing data related operations in Page Reports and RDL Reports.

In this section

Group Data

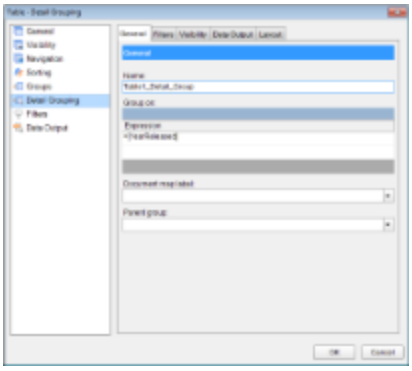
In a page report and RDL report, you can set grouping to organize data in your reports. The most common grouping scenario is to create groups by fields or expressions in a data region.
Depending on the data region you select, you can group data in one of the following ways:

- In a Table or Banded List, you can add group header and footer rows. You can also set detail grouping in the Table data region.
- In a List, you can set detail grouping.
- In a Tablix, you can add columns and rows either dynamically or manually to group data.
- In a Classic Chart , you can group data by categories or series.

To group in a data region

Detail Grouping is useful when you do not want to repeat values within the data on display in the report.

1. From the Visual Studio toolbox, drag and drop a Table data region onto the design surface.
2. With the Table selected on the report, under the Properties Window, click the **Property dialog** link to open the Table dialog > Detail Grouping page.
3. In the **Detail Grouping** page, under **Group on** enter the expression on which you want to group the data. For e.g., `=Fields!YearReleased.Value`



> **Note:** A default group name like Table1_Detail_Group1 appears under **Name**, until you set the grouping expression under **Group On**, after which you can modify the group name.

4. Under the **Document map label**, you can optionally set a label to add the item to the document map. See Document Map for further information.
5. Under **Parent group**, you can optionally set the parent group for a recursive hierarchy.
6. Click **OK** to close the dialog.

7. Drag and drop fields onto the Table data region and go to the preview tab to see grouped data.

Tip: You can filter, set page breaks or set visibility for your grouped data in the other tabs of the Table-Detail Grouping page.

To group in a FixedPage

In a Page Report, you can group your data on the fixed page. A group set on the fixed page, applies to the entire report including the controls within the report. Therefore, once you set grouping here, you may decide not to group individual data regions.

Use the following steps to understand grouping on a fixed page. These steps assume that you have already added a Page Report template, connected it to a data source and created a dataset. See Quick Start and Connect to a Data Source for further information.

Note: This topic uses the **Movie** table in the Reels database. By default, the Reels.mdb file is located in [User Documents folder]\GrapeCity Samples\ActiveReports 14\Data folder.

1. Right-click the gray area outside the design surface and select **Fixed Layout Settings** or with the fixed page selected, under the Properties Window, click the **Property dialog** link to open the FixedPage dialog.
2. On the **Grouping** page, in the General tab, under **Group on** enter the field name or expression on which you want to group the data. For example, `=Fields!YearReleased.Value`.

   Note: A default group name like FixedPage1_Group appears under **Name**, once you set the group. To modify the group name, add a field name or expression under **Group on** to enable the Name option and enter a new name.
3. Under the **Document map label** field, you can optionally set a label to add the item to the document map.
4. Click **OK** to close the dialog.
5. Drag and drop a data region, like a Table onto the design surface and set data inside its cells.
6. Go to the **Preview Tab** to view the result. You'll notice that the data appears in groups sorted according the **YearReleased** field on each report page.

Detail Grouping

Detail grouping is available in the List and Table data regions. It is useful when you do not want to repeat values within the details. When you set detail grouping, the value repeats for each distinct result of the grouping expression instead of each row of data.

For example, if you use the Movie table of the Reels database to display movie titles by year without setting detail grouping, you see each year as many times as there are movies from that year.

If you set detail grouping to `=Fields!YearReleased.Value,` each year appears only once.

Note: If the detail grouping expression you use results in a value that is distinct for every row of data, MovieID for example, you will see no difference in the results.

Recursive Hierarchies

If you want to display parent-child relationships in your data, you can create a recursive hierarchy. To do this, you need a unique ID field for the child group and an ID field for the parent group.

For example, if you have pulled data from the Reels database using the following SQL query:

SQL Query

```
SELECT EmployeePosition.*, Employee.*, Employee_1.PositionID AS ManagerPosition FROM
Employee AS Employee_1 RIGHT JOIN (EmployeePosition INNER JOIN Employee ON
```

```
EmployeePosition.PositionID = Employee.PositionID) ON Employee_1.EmployeeID =
Employee.ManagementID;
```

You can set Detail Grouping in a Table data region using the `=Fields.Item("EmployeePosition.PositionID").Value` field, and the `=Fields!ManagerPosition.Value` field as the parent group to display parent-child relationships in your data.

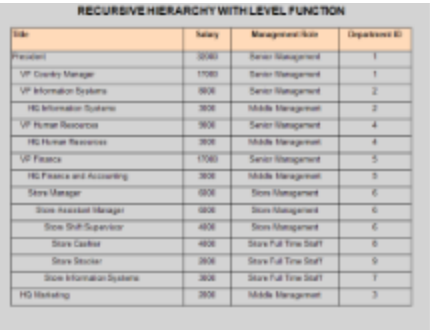
Note: You can use only one group expression when you set a parent group.

Level Function

To better visualize data in a recursive hierarchy, you can use the Level function. This function indents text and further clarifies the relationships between parent and child data. To do this, you set an expression in the Padding - Left property of the text box you want to indent.

For example, in a Table data region, for the recursive hierarchy example above, you can set the following expression in the Padding - Left property of the text box that contains the Title to indent values according to levels:

```
=Convert.ToString(2 + (Level()*10)) & "pt"
```



Set Detail Grouping In Sparklines

Applying detail grouping to a sparkline helps to visualize data clearly – the sparkline value is displayed as a set of sparklines grouped by a detail grouping value. The following steps take you through how to show trends in analyzed data when detail grouping is set with Sparklines.

These steps assume that you have already added a Page Report/RDL Report template to your project, connected it to a

data source and added a dataset. See Quick Start, Connect to a Data Source and Add a Dataset for more information.

> **Note**: This topic uses examples from the SalesByGenre table in the Reels database. The Reels.mdb file can be downloaded from GitHub: ..\Samples14\Data\Reels.mdb.

1. From the toolbox, drag a Table data region onto the report design surface.
2. With the table selected, set the following:
   - Set the DataSetName property (For example, SalesByGenre).
   - Right-click the table handle to the left of the Detail row and select **Insert Row Below** or **Insert Row Above** from the menu and add another detail row to the table.
   - Hover your mouse over the first Textbox located in the Detail row column to make the Field Selection Adorner appear. Click this adorner to display a list of available fields from the data set, select a field (For example, GenreName).

     > **Note**: This automatically places an expression in the detail row and simultaneously places a static label **Genre Name** in the header row of the same column.
3. From the toolbox, drag a Sparkline control to the second detail row of the table on your report and set the following in the Properties Window.
   - Set the **SparklineType** property to Line (by default).
   - Set the **DataSetName** property to a data set (For example, SalesByGenre).
   - Set the **SeriesValue** property by selecting <Expression...>. In the Expression Editor under Fields, expand the Fields (SalesByGenre) node and select a numeric field from the connected data set (for example, =Fields!Profit.Value). Click the OK button to accept the change.
   - Set the **LineColor** property to Gray.
   - Set the **MarkerColor** property to Blue.
4. To apply the detail grouping to the sparkline, right-click the Table data region and go to Properties Window to select the **Properties dialog** command at the bottom.
5. In the Table dialog that appears, select the **Detail Grouping**.
6. In the General tab, under **Group on**, select an expression from the drop-down list on which to group the data (for example, =Fields!GenreName.Value).
7. Click the **OK** button to close the dialog and save the changes.
8. Go to the preview tab to view the sparkline you have added to your report.

This set of sparklines display the profit value grouped by the movie genres.

Sort Data

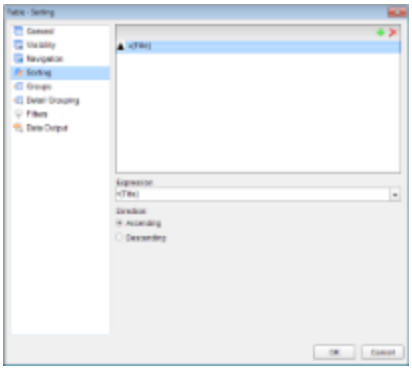
In a page report and a RDL report, you can apply sorting on a data region, grouped data or fixed page.

Use the following steps to determine how to sort your data. These steps assume that you have already added a Page Report (xml-based)/RDL Report template to your project and connected it to a data source. See Quick Start, Connect to a Data Source and Add a Dataset for more information.

To sort in a data region

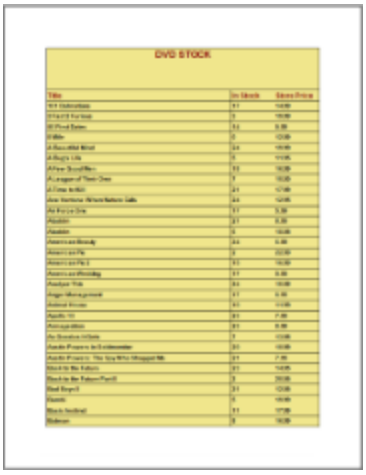
You can set the sorting expression on the Sorting page of a data region dialog.

1. Right-click the data region and select **Properties** to open the Properties window. Select the Property dialog link under properties where the commands are displayed to open the data region dialog. See Properties Window for more information.
2. In the dialog that appears, go to the **Sorting** page and click the Add(+) icon to add an empty expression to the sorting list below.
3. In the Expression field, enter the expression directly or use <Expression...> from the dropdown to open the Expression Editor and select the field on which you want to sort your data.



4. Under **Direction**, select Ascending or Descending to set the order in which to sort your data.
5. Click **OK** to close the dialog.
6. From the Report Explorer, drag and drop the field on which the sorting expression is set and go to the Preview Tab to view the result.

The following image shows the result after setting sorting in a Table data region on the **Title** field in Ascending order:

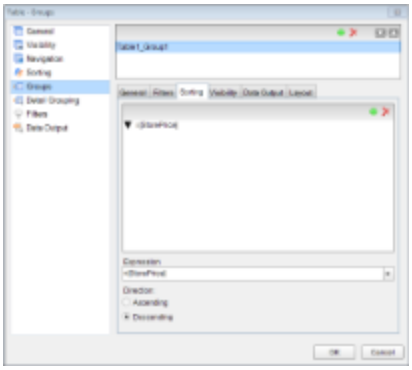


To sort on grouped data

You can sort the order of groups through the **Sorting** tab of the Groups page or the Detail Grouping page of the List data region. The following steps assume that you have already grouped data in the data region. See Group Data for further information on grouping.

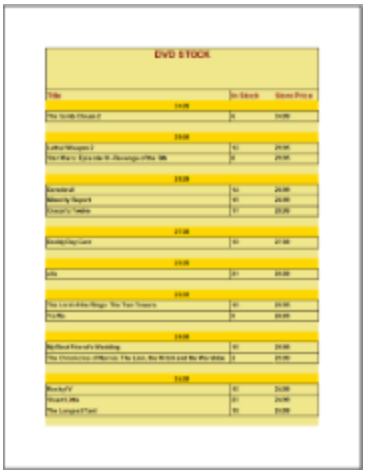
> **Note:** In a Chart data region dialog, the Sorting tab is available on the Category Groups and Series Groups pages.

1. On the Groups or the Detail Grouping page of the data region dialog, select the **Sorting** tab.
2. In the **Sorting** tab, click the Add(+) icon to add an empty expression to the sorting list.
3. In the Expression field, enter the expression directly or use <Expression...> from the dropdown to open the Expression Editor and select the field on which you want to sort your data. The expression set here should be the same as the grouping expression.



4. Under **Direction**, select Ascending or Descending to set the order in which to sort your data.
5. Click **OK** to close the dialog.
6. From the Report Explorer, drag and drop the field on which sorting is set and go to the Preview Tab to view the result.

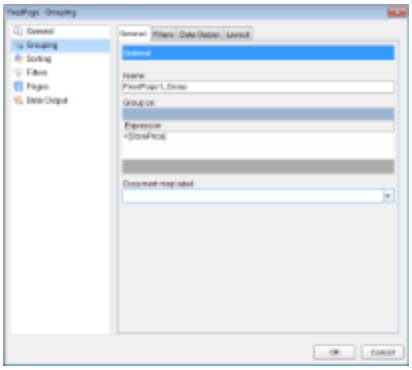
The following image shows the result on a Table data region where grouping and sorting is set on the **StorePrice** field in Descending order:



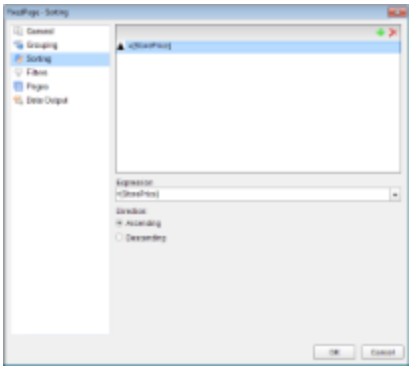
To sort on a fixed page

In a Page Report , if the fixed page is grouped on a dynamic value, you can sort the order of the groups through the Sorting page of the FixedPage Dialog. Follow the steps below to learn setting up sorting on a Fixed Page:

1. Right-click the gray area of the report and select **Fixed Layout Settings** to open the FixedPage dialog.
   OR
   Right click the report page and select Properties to open the Properties window. Select the Property dialog link under properties where the commands are displayed to open the FixedPage dialog. See Properties Window for more information on commands.
2. In the FixedPage dialog that appears, go to the **Grouping** page and in the Expression field, enter the expression directly or use <Expression...> from the dropdown to open the Expression Editor and select the field on which you want to group your data.



3. In the FixedPage dialog, now go to the **Sorting** page and click the Add(+) icon to add an empty expression to the sorting list below and in the Expression field enter the same expression you used for grouping the data.

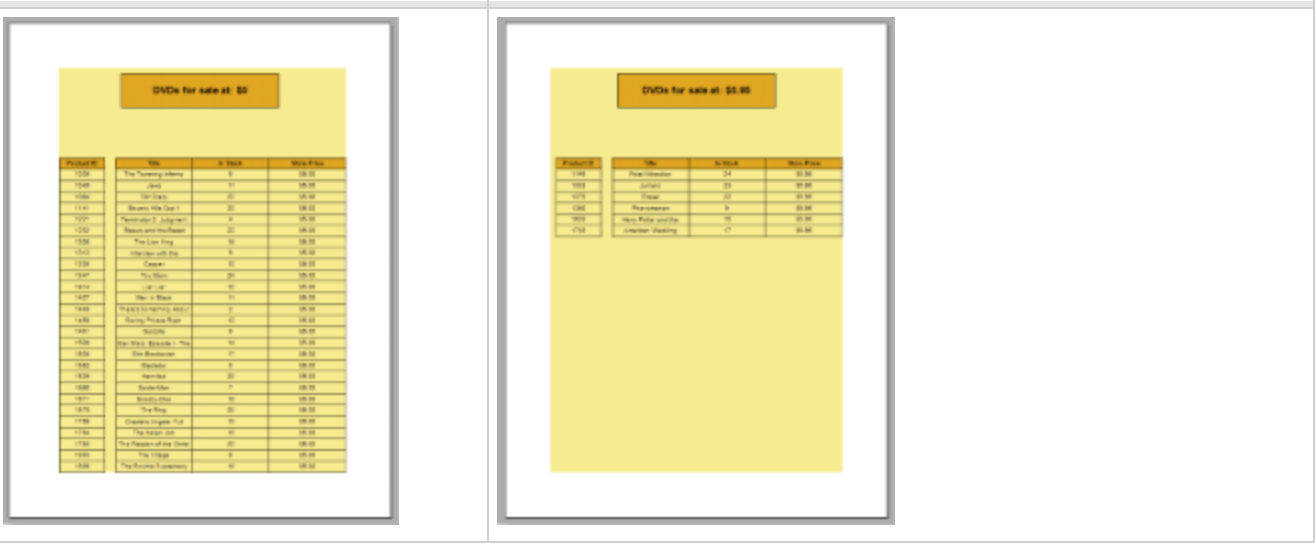


4. Under **Direction**, select Ascending or Descending to set the order in which to sort your data.
5. Click **OK** to close and apply the settings.
6. From the Report Explorer, drag and drop the field on which sorting is set and go to the Preview Tab to view the result.

> **Note:** The difference in setting sorting on a Fixed Page is that it affects every data region placed on the report layout, whereas sorting on a data region is limited to the data region only.

The following images shows the result when sorting is set on a fixed page on the **StorePrice** field in descending order:

Page 1	Page 2



Set Filters

Normally you can filter your data using parameters in a query, but if your data source does not support parameters you can use filters. You can set filters on the following:

- DataSet
- Data Region
- Groups in a Data Region

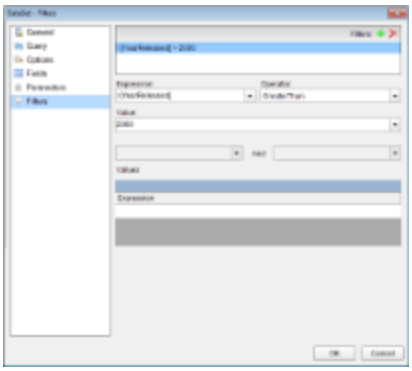
In a Page Report, you can also set filters on the page through the FixedPage dialog.

Use the following steps to create filters in a page report and RDL report. These steps assume that you have already added a Page Report/RDL Report template and have a data connection and a dataset in place. See Quick Start, Connect to a Data Source and Add a Dataset for more information.

To set filters on a DataSet

When you set a filter on a dataset, any control you add to the design surface can utilize this filtered data.

1. In the Report Explorer, right-click the DataSet node and select **Edit**.
2. In the DataSet dialog that appears, select the **Filters** page and click the Add (+) icon to add a new filter for the data set. By default, an empty filter expression gets added to the filter list.
3. Under Expression, enter an expression or use the Expression Editor to provide the expression on which to filter data. For example, `=Fields!YearReleased.Value`
4. Under Operator, select an operator from the list to decide how to compare the Expression with the Value. For example, set a GreaterThan operator on the Expression above. See Filtering for a list of available operators and their description.



5. Under Value, enter a value or set an expression using the Expression Editor with which to compare the expression results. For example, 2000 to represent the year 2000.

   The resultant filter looks like the following.
   `=Fields!YearReleased.Value > 2000`

To set filters in a Data Region

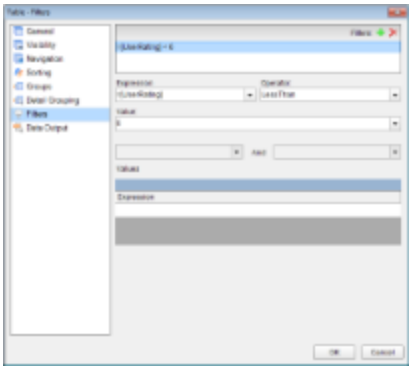
When you set a filter in a data region, you can limit the amount of data available for use inside that data region.

1. With the data region selected on the report, under the Properties window, click the **Property dialog** link. This is a command to open the corresponding data region dialog. See Properties Window for more on how to access commands.

   > **Note:** In a Classic Chart data region, right-click the data region and choose the **Chart data** option to open the Chart Data dialog.

2. In the data region dialog that appears, select the **Filters** page and click the Add (+) icon to add a new filter for the data region. By default, an empty filter expression gets added to the filter list.
3. Under Expression, enter an expression or use the Expression Editor to provide the expression on which to filter data. For example, `=Fields!UserRating.Value`
4. Under Operator, select an operator from the list to decide how to compare the Expression with the Value. For example, set a LessThan operator on the Expression above. See Filtering for a list of available operators and their

description.



5. Under Value, enter a value or set an expression using the Expression Editor with which to compare the expression results. For example, 6 to represent the Rating 6.

   The resultant filter looks like the following.
   `=Fields!UserRating.Value < 6`

To set filters on groups in a Data Region

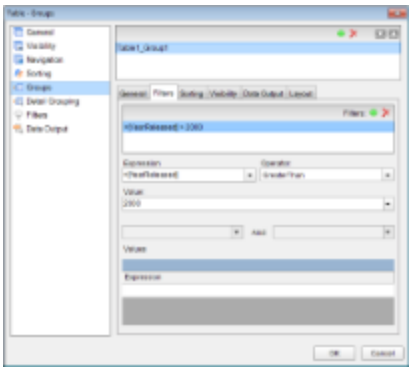
You can also set filters on grouped data in a data region. The following example uses the Table data region to show filtering on groups.

1. In the report, set grouping on a data region. For example, on a Table data region, set grouping on the `=Fields!YearReleased.Value` field. See Group Data for further details.
2. With the data region selected on the report, under the Properties window, click the **Property dialog** link. This is a command to open the corresponding data region dialog. See Properties Window for more on how to access commands.

   > **Note:** In a Classic Chart data region, right-click the data region and choose the **Chart data** option to open the Chart Data dialog.

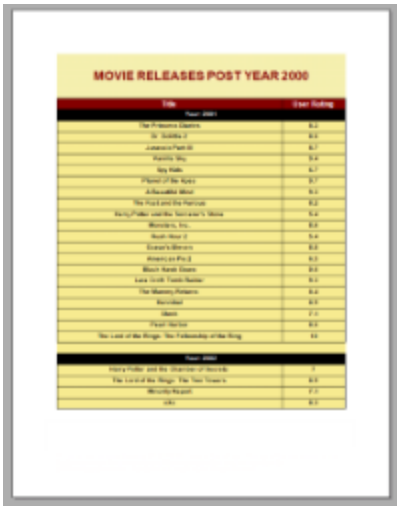
3. In the Table dialog, go to the **Groups** tab and select the Group.

4. After selecting the group, go to the **Filters** tab and click the Add (+) icon to add a new filter. By default, an empty filter expression gets added to the filter list.
5. Under Expression, enter an expression or use the Expression Editor to provide the expression on which to filter data. For example, `=Fields!YearReleased.Value`
6. Under Operator, select an operator from the list to decide how to compare the Expression with the Value. For example, GreaterThan operator set on the Expression above. See Filtering for a list of available operators and their description.



7. Under Value, enter a value or set an expression using the Expression Editor with which to compare the expression results. For example, 2000 to represent the year 2000.

   The resultant filter looks like the following.
   `=Fields!YearReleased.Value > 2000`



Page/RDL Report Scenarios

See step-by-step instructions for creating commonly used reports in a Page Layout.

In this section

Create Top N Report

A Top N Report displays details for the top results in your report. You can create this report by modifying the SQL query while creating a dataset.

The following steps demonstrate how to create a Top N report. These steps assume that you have already added a Page Report/RDL Report template to your project and connected it to a data source. See Quick Start and Connect to a Data Source for more information.

1. In the Report Explorer, right-click the data source node (DataSource1 by default) and select the **Add Data Set** option or select **Data Set** from the add button.
2. In the **DataSet** dialog that appears, go to the **Query** page and enter a query in the Query textbox in the following format : `Select Top N FieldNames From TableName`

   > **Note:** In the query above **TableName** refers to the table you want to get from the database. **FieldNames** and **N** refer to the fields you want to fetch from the table and the number of records to display from that field. Following is an example of a Top N Report SQL query : `Select Top 10 * From Movie`

3. Click the **Validate DataSet** icon at the top right hand corner above the Query box to validate the query and then click **OK** to close the dialog.
4. In the Report Explorer, expand the DataSet node and drag and drop fields onto the design surface. In an Page report, you need to place these fields inside a data region.
5. Go to the Preview tab and view the result. You'll notice only **N** number of records displaying in your report.

The following image is an example of a Top N Report displaying top 10 movie records:

Create Red Negatives Report

A Red Negatives report shows negative values in red color if those values meet the requirements set in a conditional expression. The following steps demonstrate how to create a report with negative values in red negatives.

These steps assume that you have already added a Page Report/RDL Report template to your project, connected it to a data source and added a DataSet. See Quick Start, Connect to a Data Source and Add a Dataset for more information.

1. From the Visual Studio toolbox, drag and drop a Table data region onto the design surface and place fields inside the cells of the details row.
2. In the same Table, select any cell (Textbox) that displays integer values and right click to select Properties.
3. In the Properties Window that appears, set the following expression in the **Color** property:
   `=iif(Fields!FieldName.Value < 0, "Red", "Black")`

   > **Note:** In the expression above, **FieldName** refers to field that the textbox contains. For example, if a textbox contains the Rollup field, the expression looks like: `=iif(Fields!Rollup.Value < 0, "Red", "Black")`
4. Go to the Preview tab and view the result.

The following image illustrates a report that contains negative values in red:

Create Green Bar Report

You can create a Green Bar report by alternating the background color of a data region like a Table using conditional formatting. The following steps demonstrate Follow the steps below to learn creating a Green Bar Report.

These steps assume that you have already added a Page Report/RDL Report template to your project, connected it to a data source and added a dataset. See Quick Start, Connect to a Data Source and Add a Dataset for more information.

1. From the Visual Studio toolbox, drag and drop a Table data region onto the design surface.
2. In the Table data region, click the row handle to the left of the detail row and right-click to select Properties.
3. In the Properties Window dialog that appears, set the following expression in the **BackgroundColor** property:
   `=iif(RowNumber(Nothing) Mod 2, "PaleGreen", "White")`
4. On the design surface, set fields the detail row of the table data region.
5. Go to Preview tab and view the result. You will notice that every alternate detail the report displays has a green

background.

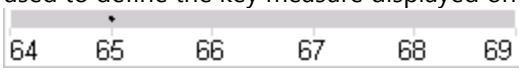
The following image shows an example of a Green Bar report:

Create a Bullet Graph

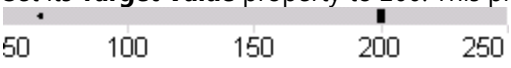
You can create a bullet graph based on aggregated data from the data source. The following steps demonstrate how to create a bullet graph.

These steps assume that you have already added a Page Report/RDL Report template to your project and connected it to a data source. See Quick Start and Connect to a Data Source for more information.

1. From the Visual Studio toolbox, drag a drop **Table** control onto the design surface.
2. From the Visual Studio toolbox, drag a **Bullet** control onto the detail row of the table and in the properties window, set its **Value** property to a numeric field (like `=Fields!SalesAmount.Value`). This Value property is used to define the key measure displayed on the graph.

3. With the Bullet control selected on the design surface:
   - Set its **Target Value** property to 200. This property defines a target for the Value to be compared to.

   - Set its **Best Value** property to 500 and the **Worst Value** property to 0. The Best Value and Worst Value properties define the value range on the graph.

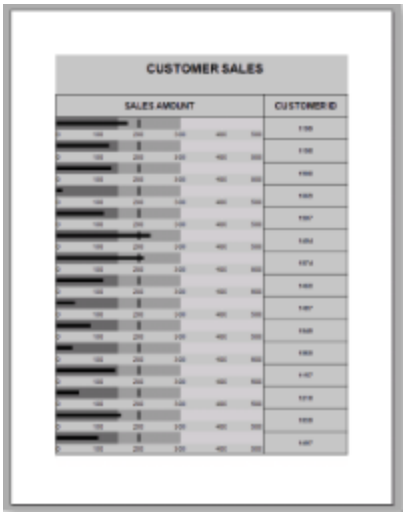
     

   - You can also optionally encode the segments on the graph as qualitative ranges indicating bad, satisfactory and good sections.
     - The **Range1Boundary** property defines a value for the bad/satisfactory boundary on the graph. Set this property to 150.
     - The **Range2Boundary** property defines a value for the satisfactory/good boundary on the graph. Set this property to 300.

   - You can also optionally define the **Interval** property for the graph value range. So, set this property to 100.

4. Go to the Preview tab to view the bullet graph you have added to your report.
   As the bullet graph is based on aggregated data, you get a stack of bullet graphs indicating the Sales Amount value for different customers.



Create a Whisker Sparkline

You can use a whisker Sparkline to render “win/loss/tie” scenarios (for example, sport statistics) or “true/false” scenarios (for example, was the sales goal met or was the temperature above average), based on the numeric data from a data set.

The bars in a whisker sparkline render below the baseline for a negative value, above the baseline for a positive value and on the baseline for a zero value, for example, in a "profit/loss/no profit, no loss" scenario.

The following steps demonstrate how to create a whisker sparkline. These steps assume that you have already added a Page Report/RDL Report template to your project, connected it to a data source and added a dataset. See Quick Start, Connect to a Data Source and Add a Dataset for more information.

> **Note**: These steps use the AccountsChartData table from the Reels database. The Reels.mdb file can be downloaded from GitHub: ..\Samples14\Data\Reels.mdb.

1. From the Visual Studio toolbox, drag a Sparkline control onto the design surface.
2. With the sparkline selected on the design surface, go to the properties window and:
   - Set the **Sparkline Type** property to **Whiskers**.
   - Set the **SeriesValue** property to a numeric field (like `=Fields!RollUp.Value`) from the connected data set.
   - Set the **FillStyle/FillColor** property to Red.
3. Go to the Preview tab to view the whisker sparkline.

Create and Use a Master Report (RDL Report)

In an RDL report, you can create a master report and apply it to any number of content reports to keep common styles in one easy-to-maintain location. See Master Reports for more information.

To design a master report

> **Note:** These steps assume that you have already added an RDL Report template and connected it to a data source. The Master Report feature is only available with RDL Reports. See the topic, Adding an ActiveReport to a Project for further information.

1. With focus on the report, from the Report Menu, select **Convert to Master Report** to create a master report.
2. Right-click the ruler area to the top or left of the report and choose **Page Header.** Repeat and choose **Page Footer**.
3. From the toolbox, drag and drop controls into the page header and footer sections. These controls appear on every page of a content report when you apply the master report. For example, a company logo image, or a textbox with the company Web site address.

4. A ContentPlaceHolder control appears in the toolbox when you convert an RDL report to a Master Report. This control defines regions where users can add content. Use the following instructions to add the ContentPlaceHolder control in the master report.

    - From the toolbox, drag and drop the ContentPlaceHolder control onto the report design surface.
    - Right-click the control and from the context menu that appears, select **Properties** to open the properties window.
    - Set the following properties for the ContentPlaceHolder control.

Property	Description
Location	To position the control with respect to the top left corner of the container.
Size	To set the control size for determining the space available to design a content report.
Text	To add instructive text for the user. E.g. "Design your content report here." This caption appears in the design view and not in the final output.

5. Save the master report locally on your system.

    1. In Visual Studio, select the report, and from the Report menu, select Save Layout. In the stand-alone designer, the option is in the File menu. You can also save the master report from File menu of the stand-alone designer.
    2. In the **Save As** dialog that appears, navigate to the location where you want to save the report and click **Save** to save the report in rdlx-master file format.

To use a master report

1. With focus on the report to which you want to apply the master report, from the Report menu, select **Set Master Report**, then **Open Local File**.
2. In the Open dialog that appears, navigate to the location where you saved the master report and open it. The master report layout is applied to the content report with all areas locked except for the region with the ContentPlaceHolder, which is available for use.

Merge Multiple Reports

You can merge or combine multiple Page and RDL reports into one report. Merging is performed by using **ReportCombiner ('ReportCombiner Class' in the on-line documentation)** class which adds the reports as subreports. The reports are merged one after the another, in the order in which they are added. To correctly combine reports, you should use reports with the same layouts: for **Page** reports, you need to set equal margins for all reports and for **RDL** reports, you need to set equal margins and width for all reports.

You can use the **BuildReport ('BuildReport Method' in the on-line documentation)** method to utilize all the features of the **PageReport** class. You can also insert page breaks and specify the gap between two reports when merging. By default, the gap of 1 inch is added between the reports.

The following code combines three reports. You need to add **GrapeCity.ActiveReports.Core.Rendering** assembly to your project. The code also exports the merged report in PDF format, so you need to add **GrapeCity.ActiveReports.Export.Pdf** assembly.

To write the code in Visual Basic.NET

Visual Basic.NET

```
Dim combiner = New GrapeCity.ActiveReports.ReportsCore.Tools.ReportCombiner()

Dim r1 = New GrapeCity.ActiveReports.PageReport()
r1.Load(New System.IO.FileInfo("c:\temp\Report1.rdlx"))

Dim r2 = New GrapeCity.ActiveReports.PageReport()
r2.Load(New System.IO.FileInfo("c:\temp\Report2.rdlx"))

Dim r3 = New GrapeCity.ActiveReports.PageReport()
r3.Load(New System.IO.FileInfo("c:\temp\Report3.rdlx"))

combiner.AddReport(r1)

Dim options = New GrapeCity.ActiveReports.ReportsCore.Tools.LocationOptions
options.Gap = "5in" 'adds a 5 inch gap from the first report. By default this gap is 1
inch.
options.PageBreakBefore = True 'adds a page break.

combiner.AddReport(r2, options)
combiner.AddReport(r3)

'PDF Rendering extension
Dim pdfRe = New GrapeCity.ActiveReports.Export.Pdf.Page.PdfRenderingExtension()
Dim provider = New GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider(New
System.IO.DirectoryInfo("c:\temp\"), "CombinedReport")

Viewer1.LoadDocument(combiner.BuildReport().Document)    'load combined report in viewer
combiner.BuildReport().Document.Render(pdfRe, provider) 'export combined report in PDF
format
```

To write the code in C#

C# code

```
var combiner = new GrapeCity.ActiveReports.ReportsCore.Tools.ReportCombiner();

var r1 = new GrapeCity.ActiveReports.PageReport();
r1.Load(new System.IO.FileInfo(@"c:\temp\Report1.rdlx"));

var r2 = new GrapeCity.ActiveReports.PageReport();
r2.Load(new System.IO.FileInfo(@"c:\temp\Report2.rdlx"));

var r3 = new GrapeCity.ActiveReports.PageReport();
r3.Load(new System.IO.FileInfo(@"c:\temp\Report3.rdlx"));

combiner.AddReport(r1);
combiner.AddReport(r2, new LocationOptions() { PageBreakBefore = true, Gap = "5in" });
//adds second report after a page break and a 5 inch gap from the first report. By
default this gap is 1 inch.
combiner.AddReport(r3);

//PDF Rendering extension
var pdfRe = new GrapeCity.ActiveReports.Export.Pdf.Page.PdfRenderingExtension();
var provider = new GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider(new
System.IO.DirectoryInfo(@"c:\temp\"), "CombinedReport");

viewer1.LoadDocument(combiner.BuildReport().Document);   //load combined report in
viewer
combiner.BuildReport().Document.Render(pdfRe, provider); //export combined report in PDF
format
```

The merged reports can be modified in many ways as described below.

To add reports at particular index

If you want to add a report r4 after the first report r1, use the following code with index '1':

C#

```
combiner.Insert(1, r4, new LocationOptions());
report = combiner.BuildReport();
```

VB.NET

```
combiner.Insert(1, r4, New GrapeCity.ActiveReports.ReportsCore.Tools.LocationOptions())
report = combiner.BuildReport()
```

To add a list of reports

C#

```
combiner.AddRange(new PageReport[] {r1, r2, r3, r4 }, new LocationOptions())
report = combiner.BuildReport();
```

VB.NET

```
Dim reports As IEnumerable(Of GrapeCity.ActiveReports.PageReport) = {r1, r2, r3, r4}
combiner.AddRange(reports, New
GrapeCity.ActiveReports.ReportsCore.Tools.LocationOptions())
report = combiner.BuildReport()
```

To delete report(s)

If you want to delete a report in first position, use the following code with index '0':

C#

```
combiner.RemoveAt(0);
report = combiner.BuildReport();
```

VB.NET

```
combiner.RemoveAt(0)
report = combiner.BuildReport()
```

Similarly, if you want to delete report at second position, use index '1'.

If you want to delete all instances of report r2, use the following code:

C#

```
combiner.RemoveAll(r2);
report = combiner.BuildReport();
```

VB.NET

```
combiner.RemoveAll(r2)
report = combiner.BuildReport()
```

Note:

- If the reports being merged are of different sizes, the page size of the first report is applied to all the pages of combined report.
- If the reports being merged are of different paper orientations, the paper orientation of the first report is applied to all the pages of combined report.

Interactivity

Learn to perform interactive tasks in Page Reports and RDL Reports with quick how-to topics.

In this section

Add Parameters
: Learn how to add parameters to page reports or RDL reports

Add Hyperlinks
: Learn how to add hyperlinks

Add Bookmarks
: Learn how to add bookmarks to move to another area in reports

Create a Drill-Down Report
: Learn how to create a drill-down report

Set a Drill-Through Link
: Learn how to use drill-through links to connect one report to another detailed report

Allow Users to Sort Data in the Viewer
: Learn how to allow the end-user to sort data in the Viewer

Add Parameters

You can add parameters to a page report/RDL report to allow users to select the data to display, or to use in creating drill-through reports.

To add a parameter

1. In the Report Explorer, right-click the **Parameters** node and select **Add Parameter**. The Report Parameters dialog appears.
2. On the **General** tab of the dialog, set the name, data type and prompt text for the parameter. For example:
    - Name: MPAA
    - Data type: String
    - Text for prompting users for a value: Enter value

    Select out of the checkbox options to allow null values, multivalues, blank value, multiline values or set hidden parameters.

3. On the **Available Values** tab, you can select **From query** populate a list from the data set from which users can select a value. Alternatively, you can select **Non-queried** to enter your own values.
4. On the Default Values tab, you can provide default values to use if the user does not select a value. This is useful when you are creating a hidden parameter.
5. Click **OK** to save the parameter. The new parameter appears in the Report Explorer under the Parameters node.
6. From the Report Explorer, drag the parameter to report design surface to create a TextBox that is bound to the parameter. When you run the report, the value that the user supplies in the prompt dialog displays in the bound TextBox on the report.

For a step by step description of adding parameters in different scenarios look at the following pages:

Add a Multi-Value Parameter
: Learn how to create a multi-value parameter.

Add a Cascading Parameter
: Learn how to create cascading parameters where one parameter value is dependent on the selection of another.

Set a Hidden Parameter
: Learn how to set a hidden parameter to allow data to fetched using the parameter value without prompting the user.

Add a Multi-Value Parameter

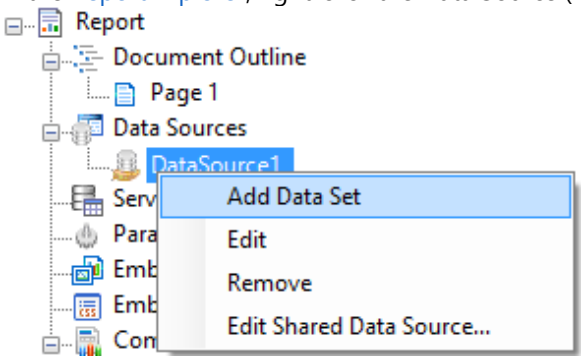
In a page report or an RDL report, you can create a multi-value parameter by selecting the Multivalue option. In a multi-value parameter, you can choose a few options from the list or simply choose 'Select all' to select all options.
If there are large number of options to choose from, choosing 'Select all' option creates an SQL query too long for an SQL Command to run. In such a case, you can specify a value to the multi-value parameter in **Value for 'Select All'** option.

The following procedures take you through a step by step process of how to create a multi-value parameter and specify a value for selecting all options from the list. These steps assume that you have added a Page Report/RDL Report template to your report and have a data connection in place. See Quick Start and Connect to a Data Source for further information.

> **Note:** This topic uses the Products table from the NWind database. The NWIND.mdb file can be downloaded from GitHub: ..\Samples14\Data\NWIND.mdb.

To create a dataset to populate the parameter values

1. In the Report Explorer, right-click the Data Source (DataSource1 by default) node and select **Add Data Set**.



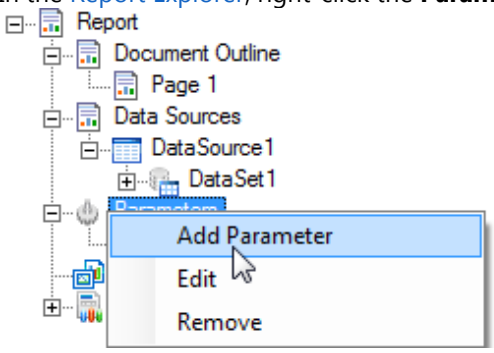
2. In the DataSet dialog that appears, select the **Query** page.
3. Enter an SQL query like the following into the **Query** text box

```
select distinct productName from Products
```

4. Click the **OK** to close the dialog. You see the data set, **DataSet1**, and the field, **productName**, in the Report Explorer.

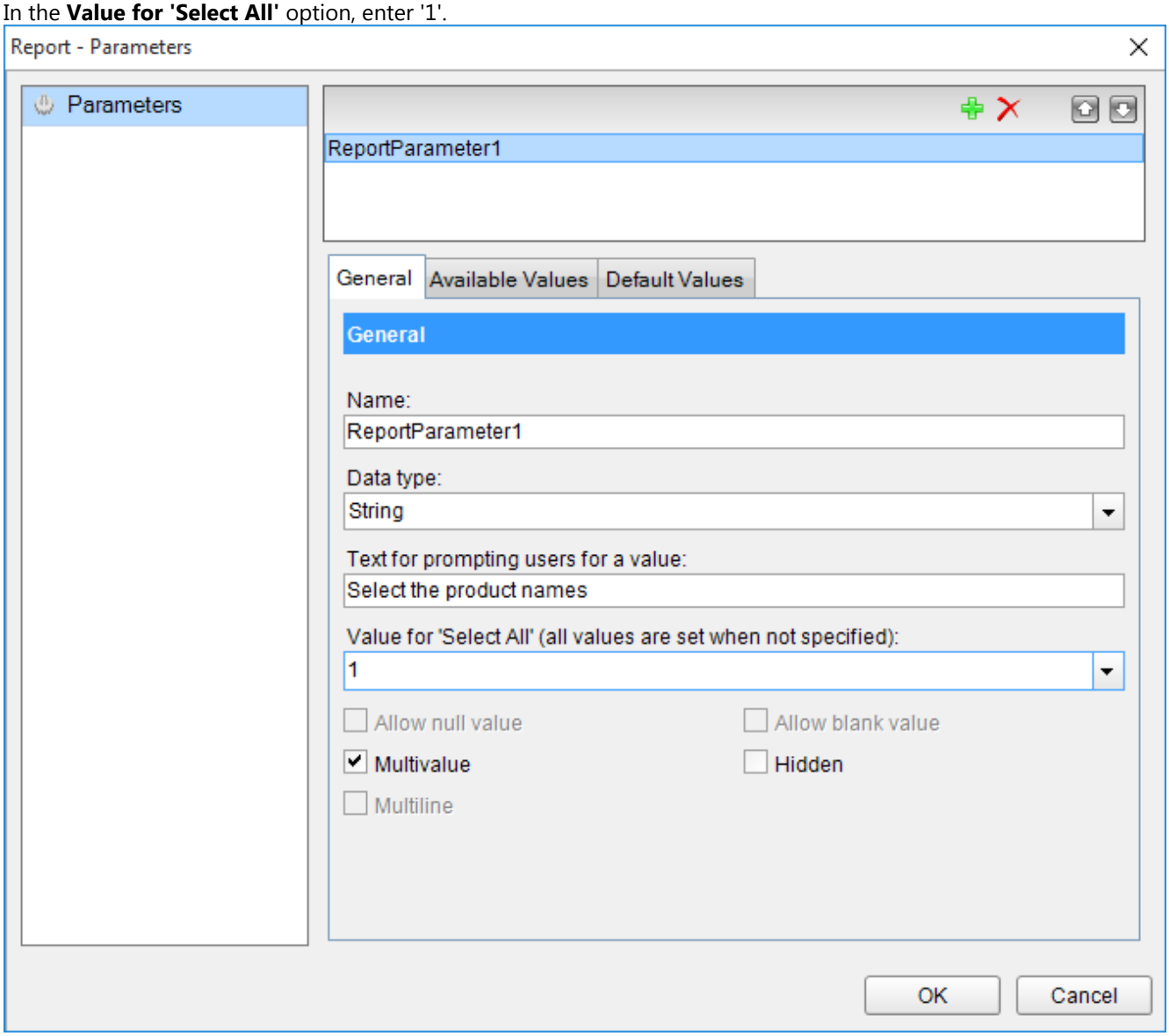
To add a Report Parameter

1. In the Report Explorer, right-click the **Parameters** node and select **Add Parameter**.



2. In the **Report - Parameters** dialog that appears, add a name for the parameter, **ReportParameter1**.
3. Ensure that the **Data type** matches that of the field (String for ProductName).
4. Enter some text in the **Text for prompting users for a value** field.

5. Select the check box next to **Multivalue** to allow users to select more than one item from the list.
6. In the **Value for 'Select All'** option, enter '1'.



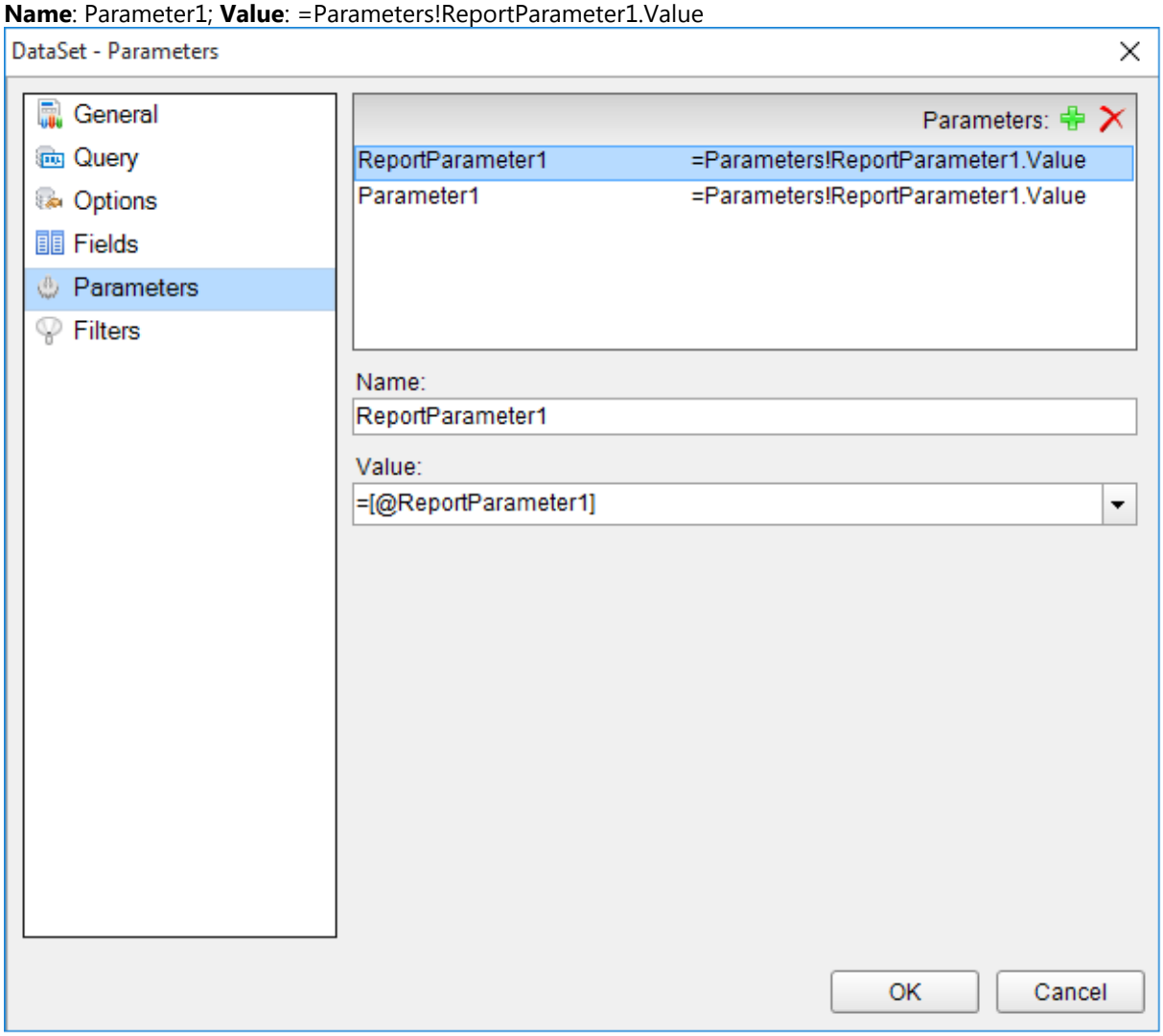
To provide a list of values for the Report Parameter

1. In the **Report - Parameters** dialog, go to the Available Values tab and select the **From query** radio button.
2. Under the **Dataset** field, select the dataset created previous steps (DataSet1).
3. Under the **Value** and **Label** fields, select the field productName.
4. Click the **OK** to close the dialog and add the parameter to the collection.

To add a dataset with a parameter

1. In the Report Explorer, right-click the Data Source (DataSource1) node and select **Add Data Set**.
2. In the DataSet dialog that appears, on the **Parameters** page, click the Add (+) icon above the parameters list and add the following to the dataset to provide values for the parameters we add to the query in step 3 below.
   Name: ReportParameter1; **Value**: =Parameters!ReportParameter1.Value

Name: Parameter1; **Value**: =Parameters!ReportParameter1.Value



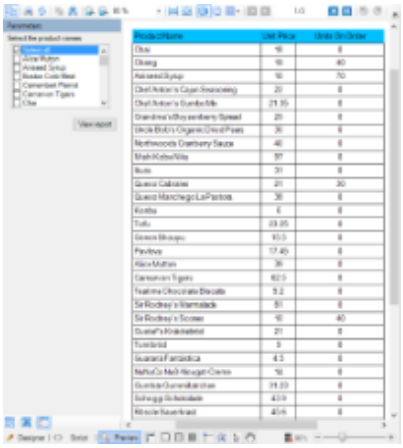
3. On the **Query** page, enter a SQL query like the following in the **Query** text box:

   ```
   SELECT * FROM products where ProductName in (?) OR '1' in (?)
   ```

   At run time, this query matches the selected product name and fetches data accordingly. If the user chooses 'Select all' (for which we have specified value '1'), then query after 'OR' is evaluated and data is fetched for all products.
4. Click the **Validate DataSet** icon to validate the query and to populate the Fields list.
5. Click the **OK** to close the dialog. You see the data set, **DataSet2**, and the fields in the Report Explorer.

To view the report

Place a control like a Table onto the design surface and add fields to it. View the report in the preview tab and see the Parameters in the sidebar with **Select all** option at the top.



Note: If the **Available Values** (queried or non-queried) for a parameter contain only some values from database and **Select all** value is specified for the parameter, then on previewing report, selecting 'Select all' checkbox shows all records from the database instead of only those present in the parameter. For example, if in database there are four records and in Available values there are only two records with Select all value specified, then on previewing report and selecting 'Select all' checkbox, all four records are shown instead of only two.

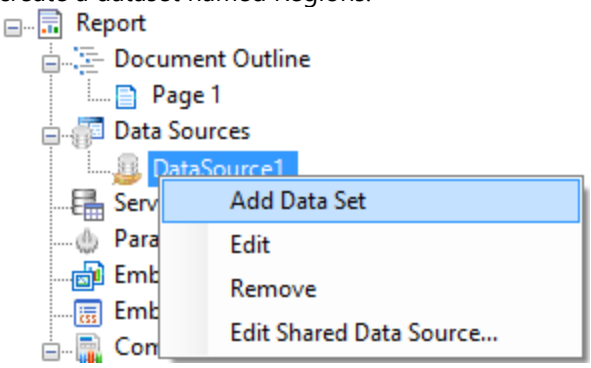
Add a Cascading Parameter

When a parameter's value list depends on the value of another parameter, the report collects the required parameter value and uses it to create the value list for the second parameter. This cascade of parameter values is sometimes also called dependent or hierarchical parameters.

Use the following instructions to create your own cascading parameters. These steps assume that you have added a Page Report/RDL template to your report and have a data connection in place. See Quick Start and Connect to a Data Source for further information. Also refer to Add a Dataset before reading this topic.

Note: This topic uses the Reels database. The Reels.mdb file can be downloaded from GitHub: ..\Samples14\Data\Reels.mdb.

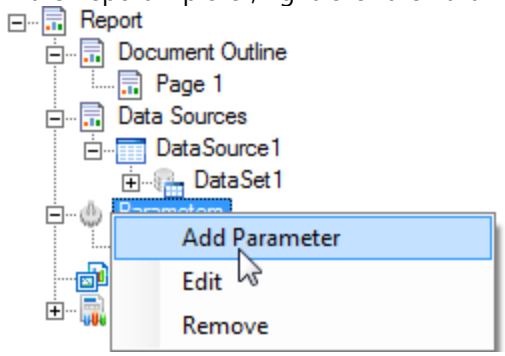
1. In the Report Explorer, right-click the Data Source (DataSource1 by default) node and select **Add Data Set** to create a dataset named Regions.



2. On the Query page of the DataSet Dialog, use the following SQL Query to fetch data from the Regions table.

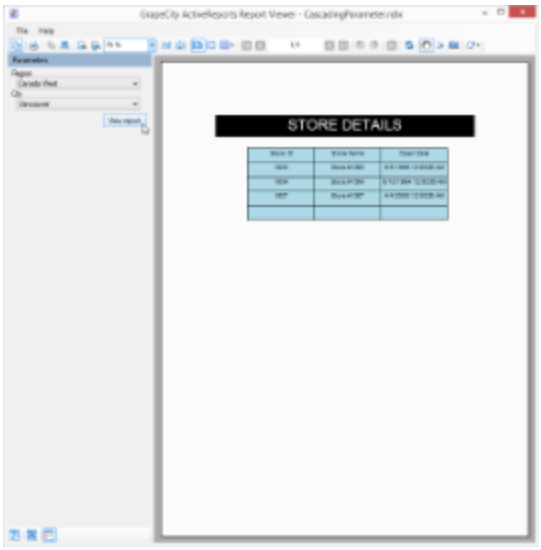
```
SELECT RegionID, Region FROM Regions
```

3. Click **OK** to close the Regions DataSet dialog.
4. Follow step 1 to create another dataset named Districts and on the Parameters page of the DataSet Dialog, click the Add(+) icon to add a parameter named **Region** with the value set to:
   `=Parameters!Region.Value`
   This parameter is added to the Report Parameters collection later.
5. In the Districts dataset dialog, on the Query page, add the following SQL query to fetch data from the Districts table. This query depends on the Region parameter.
   `SELECT DistrictID, District FROM Districts WHERE Region = ?`
6. Click **OK** to close the Districts DataSet dialog.
7. Follow step 1 and create another dataset named StoreNames and on the Parameters page of the DataSet Dialog, click the Add(+) icon to add a parameter named **DistrictID** with the value set to:
   `=Parameters!DistrictID.Value`
   This parameter is added to the Report Parameters collection later.
8. In the StoreNames dataset, on the Query page, add the following SQL query to retrieve data for the selected region from the selected district. This query depends on the DistrictID parameter.
   `SELECT StoreID, StoreName, OpenDate FROM Store WHERE NOT StoreID = 0 AND DistrictID = ?`
9. Click **OK** to close the StoreNames DataSet dialog.
10. In the Report Explorer, right-click the **Parameters** node and select **Add Parameter**



11. In the **Report - Parameters** dialog that appears, add a parameter named **Region** with an Integer data type. On the Available Values tab, select **From query** and set the dataset to Regions, the value field to RegionID, and the label field to Region.
12. Click **OK** to close the Report - Parameters dialog.
13. Follow the same process as steps 10 and 11 to add a second parameter named **DistrictID** with an Integer data type. On the Available Values tab, select **From query** and set the dataset to Districts, DistrictID for the value field, and District for the label field.
14. From the Visual Studio toolbox, drag and drop a Table data region (or any other data region) onto the design surface, and drag the **StoreID**, **StoreName** and **OpenDate** fields onto the table details row.
15. Click the Preview Tab to view the result.

Notice that the two drop down lists, for regions and districts appear in the Parameters sidebar while the second drop down list remains disabled until a region is selected. Click the **View Report** button to see the StoreID, StoreName and OpenDate values returned for the selected region and district.



Note: In a Page Report, when you have multiple datasets in the report, you need to set the **DataSet** property on the **General** tab of the FixedPage dialog in order to specify which dataset is used to display data in the report.

Set a Hidden Parameter

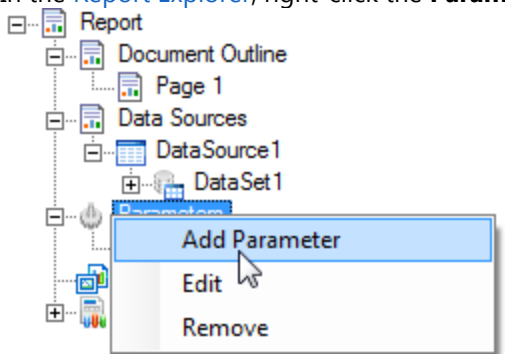
If you want to run a report without prompting the user for a value at run time, you need to set a default value for each parameter. The report collects the required parameter value from the default value and uses it to generate the report.

Default values can be queried or non-queried. A non-queried default value can be a static value or an expression. A queried default value is a field value from a dataset.

Use the following instructions to create your own hidden parameters. These steps assume that you have added a Page Report/RDL Report template to your report and have a data connection in place. See Quick Start and Connect to a Data Source for further information. Also refer to Add a Dataset before reading this topic.

Note: This topic uses the DVDStock table in the Reels database. The Reels.mdb file can be downloaded from GitHub: ..\Samples14\Data\Reels.mdb.

1. In the Report Explorer, right-click the **Parameters** node and select **Add Parameter**.



2. In the **Report - Parameters** dialog that appears, add a parameter named **StorePrice** with an Integer data type. Click the checkbox next to **Hidden** to hide the parameter UI at run time.
3. On the Default Values tab, select **Non-queried** and click the Add(+) icon to add an empty expression for the value.

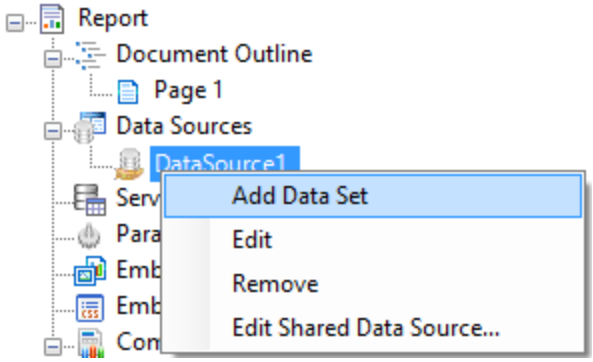
Note: When you use **From query** to provide a default value, only the first returned row value is used as the

> default value.

4. In the **Value** field enter 5 and click **OK** to close the Report - Parameters dialog.

> **Note**: When adding multiple default values, in the Report - Parameters dialog, General tab, check the **Multivalue** check box, otherwise the report collects only the first default value from the list and uses it to generate the report.

5. In the Report Explorer, right-click Data Source (DataSource1 by default) node and select **Add Data Set** to create a dataset.



6. In the DataSet Dialog that appears, on the Parameters page, click the Add(+) icon to add an empty expression for the parameter.
7. In the Name field, enter the same parameter name (**StorePrice**) you had added in the steps above and set its value to:
   `=Parameters!StorePrice.Value`
8. On the Query page of the DataSet Dialog, use the following SQL query to fetch data from the DvDStock table.
   `SELECT * FROM DvDStock WHERE StorePrice IN (?).`
9. From the Visual Studio toolbox, drag and drop a Table data region (or any other data region) onto the design surface, and from the Report Explorer, drag the **Title**, **StorePrice** and **In Stock** fields onto the table details row.
10. Click the Preview Tab to view the result.

Notice that the report collects the required parameter value from the default value (i.e. 5) and uses it to display the list of Movie DVDs with Store Price $5.

Add Hyperlinks

In a page report or a RDL report, you can set hyperlinks in the TextBox or Image controls to access a Web page from your report. You can also set hyperlinks on Map layer data elements. These hyperlinks open in the default browser of the system.

To add a hyperlink in the Textbox or Image control

1. Add a Textbox or Image control to the design surface.
2. With the control selected, under the Properties window, click the **Property dialog** link to open the respective control's dialog and go to the Navigation page.
   OR
   With the control selected, go to the Properties window and click the ellipses near the **Action** property to open the Navigation page in the dialog.
3. On the Navigation page, select the **Jump to URL** radio button to enable the field below it.
4. Type or use the expression editor to provide a valid Web page address. For example, https://www.grapecity.com/activereportsnet
5. Click **OK** to close the dialog.
6. In the Properties window, enter text in the **Value** property of the respective control to set the display text for the Web page hyperlink. For example, GrapeCity GrapeCity.

To add a hyperlink on Map layer elements

Map layer elements like point, polygon and line provides you a functionality to set hyperlinks on them to access a Web page from your report.

1. On the design surface, click the map until the map panes appear.
2. In the layers pane, right click the layer in use and select **Edit**.
3. In the selected layer's dialog that appears, go to the **Navigation** page.
4. On the Navigation page, select the **Jump to URL** radio button to enable the field below it.
5. Type or use the expression editor to provide a valid Web page address. For example, https://www.grapecity.com/activereportsnet
6. Click **OK** to close the dialog.

Add Bookmarks

A bookmark link is similar to a hyperlink, except that it moves the viewer to another area in the report instead of opening a web page. You can add bookmarks in a two-step process:

- Identify the place (target control) where you want to allow a user to jump to with the help of a Bookmark ID.
- Share that Bookmark ID with another control that links to the target control.

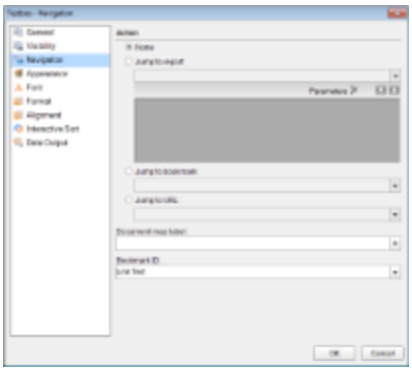
Use the following steps to create a Bookmark ID on a Textbox control and create a bookmark link on another Textbox control at the bottom of the page.

These steps assume that you have already added a Page Report/RDL Report template to your project. See Quick Start for further details.

To add a Bookmark ID on a control

Bookmark ID is like a URL that provides information required by the report viewer to locate the report control. You need to provide a Bookmark ID for any control to which you want to allow users to jump to via a Bookmark Link.

1. From the Visual Studio toolbox, drag and drop a Textbox control onto the design surface.
2. Select the Textbox to view its properties in the Properties window and enter any text in the **Value** property (For e.g., Top).
3. Click the **Property dialog** link below the Properties window to open the Textbox dialog.
4. In the TextBox dialog that appears, select the **Navigation** page and in the **Bookmark ID** field enter text like *Link Text*.



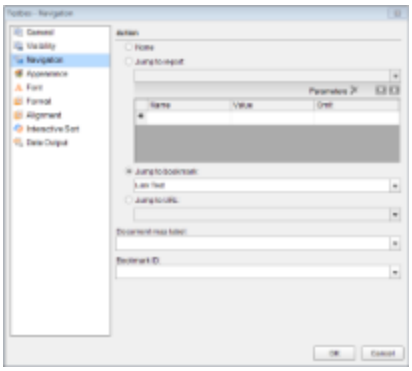
5. Click **OK** to close the dialog.

Tip: You can also set the Bookmark ID through the **Bookmark** property in the Properties window.

To set a bookmark link

Bookmark Link is a simple link you create to jump to the location where the Bookmark ID is set.

1. From the Visual Studio toolbox, drag and drop another Textbox control onto the design surface. Place it at the bottom of the page for this example.
2. Select the Textbox to view its properties in the Properties window and in the **Value** property enter *Go to Top*.
3. Click the **Property dialog** link below the Properties window to open the Textbox dialog.
4. In the Textbox dialog that appears, click on the **Navigation** page and select the **Jump To Bookmark** radio button to activate it.
5. Under **Jump To Bookmark**, enter the same text (i.e. *Link Text)* you assigned as Bookmark ID in the steps above.



6. Click **OK** to close the dialog.

7. Go to the Preview Tab, and click *Go to Top*.

You move to the top of the page where the Bookmark ID was set on the control.

> **Tip:** You can also access the Navigation page of a control to set the bookmark link through the ellipsis button next to the **Action** property in the Properties window.

Create a Drill-Down Report

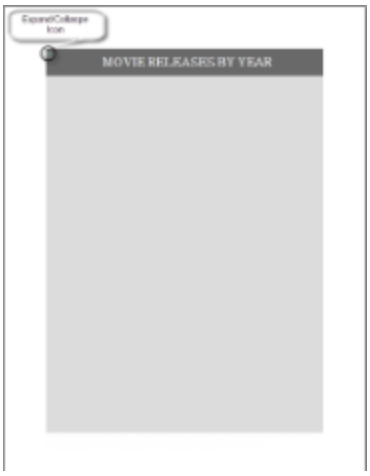
In a page layout, you can set up data regions, report controls, table rows, and tablix row and column groups to collapse, so that users can drill down into the data they choose to view.

In order to collapse an item, you use the **Visibility** settings available in the Properties Window or in the control dialog. You set the initial visibility of the report controls to Hidden and allow the user to toggle them by clicking other report controls (usually a TextBox).

When the report is initially displayed at run-time, the toggle items display with plus sign icons that you can click to display the detail data. Use the following steps to set a drill-down link.

1. From the Visual Studio toolbox, drag and drop a **TextBox** control and a **Table** data region onto the report design surface.
2. Place the TextBox control such that it appears as a header on your report.
3. From the Report Explorer, expand your data set and drag fields and place them inside the detail row of the Table data region. Expressions for these fields appear in the detail row, and labels appear in the table header row.
4. With the Table data region selected on the design surface, under the Properties window, click the Property dialog link. This is a command to open the respective control's dialog. See Properties Window for more on how to access commands.
5. In the Table dialog that appears, go to the Visibility page, change the **Initial visibility** to **Hidden**, and select the check box next to **Visibility can be toggled by another report control**.
6. From the drop-down list that appears, select the TextBox that you added in step 1. The TextBox is now used to toggle items in the Table and show detail data.
7. Click **OK** to save the changes.

When you view the report, the Textbox displays an Expand/Collapse icon to its left.

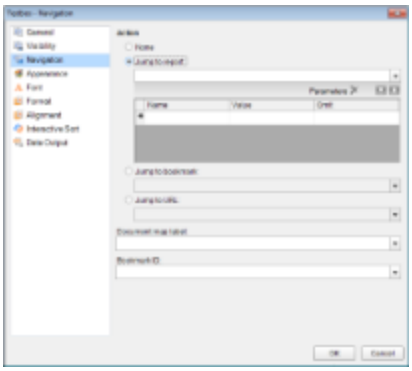


Click the icon to view the hidden data.

Set a Drill-Through Link

In a page report or a RDL report, you can use drill-through links to connect one report to another detailed report. Follow the steps to set a drill-through link:

1. On the design surface, select a report control (like a TextBox) on which you want set the link and under the Properties window, click the Property dialog link. This is a command to open the TextBox dialog. See Properties Window for more on how to access commands.
2. In the control dialog that appears, go to the **Navigation** page and under **Action**, select the **Jump to report** radio button. Doing this activates the fields below it.



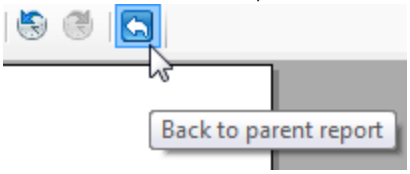
3. Under the **Jump to Report** field, enter the name of the report (like BasicReport.rdlx) that you want to navigate to on clicking the drill-through link. You can also use expressions to create drill-through links.

   > **Note:** In the **Jump to Report** field, enter just the report name if the targeted report is in the same directory as the parent report. Or you can enter a relative path to the report. Use the Custom Resource Locator to jump to a report in your connected database.

4. After setting the detail report to drill-through, on the **Navigation** page, under **Parameters**, you can optionally enter a valid parameter **Name** and **Value** to pass to the detail report. This value must evaluate to a valid value for the parameter. By setting parameters you can jump right to the desired information. For example, if your summary report contains a list of invoice numbers and delivery dates for each customer, you could use a drill-through link with the invoice number as the parameter to allow the user to jump to the relevant invoice.

   > **Caution:** The Parameter Name must exactly match the name of the parameter in the detail report. If any parameter is spelled differently, capitalized differently, or if an expected parameter is not supplied, the drill-through report fails.

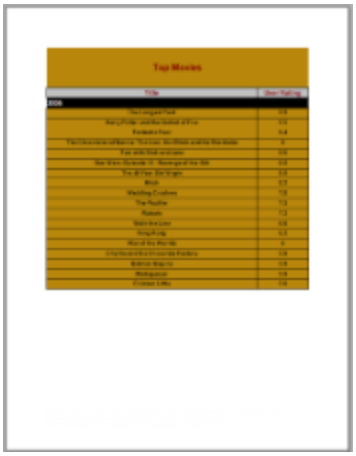
5. Go to the preview tab and click the drill-through link to navigate to the targeted report.
6. On the Viewer toolbar, click the **Back to Parent Report** button to return to the main report.



The following images show a simple drill-through link set on a list displaying years. Click any year to drill-through to a report that contains top movies in that year.

Report With a Drill-Through Link

Target Report



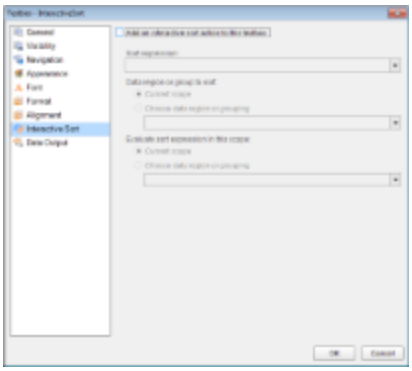
Allow Users to Sort Data in the Viewer

In a page report or a RDL report, you can allow the end-user to sort columns of data in the Viewer by setting interactive sorting on a TextBox control within a data region. Follow the steps below to set interactive sorting in a TextBox control.

To set interactive sort properties on a TextBox

These steps assume that you have already added a Page Report (xml-based)/RDL Report template to your project and connected it to a data source and a data set. See Connect to a Data Source and Add a Dataset for further information.

1. From the toolbox, drag a Table data region onto the report.
2. From the Report Explorer, drag fields into the detail row of the table. Labels appear in the table header row, and expressions appear in the detail row.
3. Click to select a TextBox in the header row on which you want to allow users to sort, and in the Commands section at the bottom of the Properties Window, click the **Property dialog** command.
4. In the TextBox dialog that appears, select the **Interactive Sort** page.



5. Select the checkbox next to **Add an interactive sort action to this textbox** and enable the other properties on the page.
6. Under **Sort expression**, drop down the list and select the expression containing the value of the field on which you want to provide sorting.

   > **Note**: Under **Data region or group to sort**, and **Evaluate sort expression in this scope**, you can optionally choose a different a scope from the **Choose data region or grouping** drop down list.

7. Click **OK** to close the dialog and accept the changes.

When you preview the report, you can see a sort icon next to the TextBox control, the **User Rating** header in this case.

You can click the icon to sort in descending order. The icon changes to an up arrow that you can click to sort ascending.

Common Tasks

Learn to perform common tasks in Page Reports and RDL Reports with quick how-to topics.

In this section

Add Items to the Document Map
: Learn how to add report controls, groups and detail groups to the Document Map

Change Page Size
: Learn how to can change the size of your page

Add Code to Layouts Using Scripts
: Learn how to add custom code in your expressions to extend report capabilities.

Freeze Rows and Columns (RDL Report)
: Learn how to Freeze Rows and Columns in RDL Report

Add Page Numbers
: Learn how to add page numbers in Page reports or RDL reports

Add Page Breaks in RDL (RDL Report)
: Learn how to add Page Breaks in RDL reports

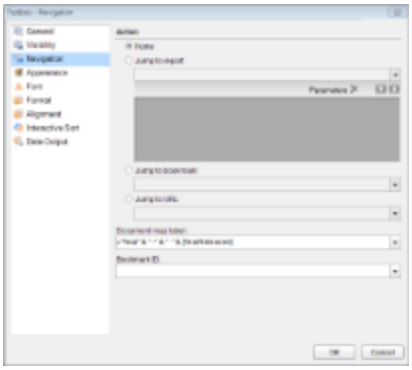
Add Items to the Document Map

In a page report or a RDL report, use the following steps to add report controls, groups and detail groups to the document map.

To add a control to the Document Map

Using Document Map Label or Label Property

1. On the design surface, select a control you want to add to the Document map and right-click to choose Properties from the context menu.
2. In the command section of the Properties Window, click Property dialog. This is a command to open the control's dialog. See Properties Window for more information on how to access commands.
3. In the dialog that appears, go to the **Navigation** page and under the **Document map label**, enter a text or an expression representing the control in the Document map.



Alternatively,

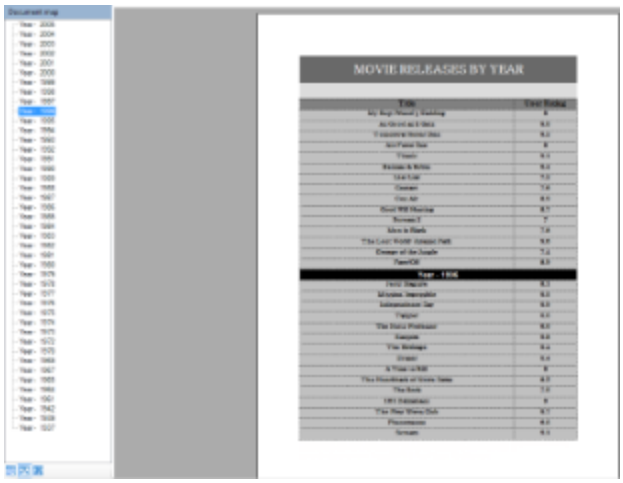
1. On the design surface, select the report control you want to add to the Document map and right-click to choose Properties from the context menu.
2. In the Properties window that appears, enter a text or an expression in the **Label** property to represent the report control in the Document map.

Using HeadingLevel Property

1. On the design surface, select the report control you want to add to the Document map and right-click to choose Properties from the context menu.
2. In the Properties window that appears, set the **HeadingLevel** property of the report control.

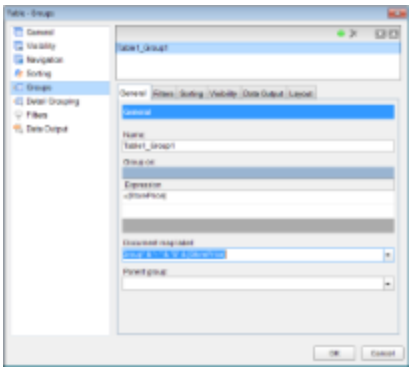
> **Note** The HeadingLevel property defines the DocumentMap level for that control.

Go to the preview tab or the Viewer to view the document map.

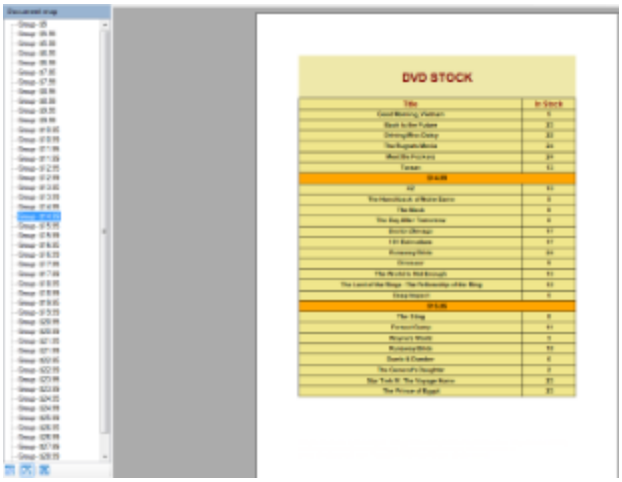


To add a group or a detail group to the Document Map

1. Open a report that contains a group. See Group Data for further information.
2. On the design surface, select the data region on which grouping or detail grouping has been set and go to the command section which appears below the Properties Window.
3. Click **Property dialog** to open the data region dialog. See Properties Window for more information on how to access commands.
4. In the dialog that appears, go to the **Grouping** or **Detail Grouping** page, and under the **Document map label**, enter a text or an expression representing the group or detail group in the Document Map.



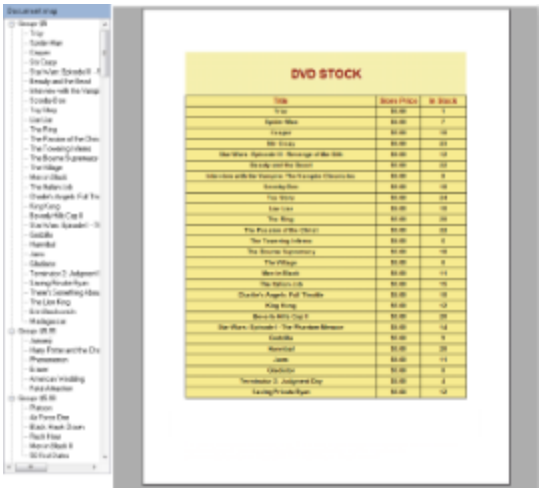
Go to the preview tab or the Viewer to view the document map.



To create a hierarchical Document Map

1. Open a report that contains a data region like a Table.
2. With the data region selected, right-click to open the context menu and select **Properties**.
3. In the Properties Window that appears, select the **Property dialog** command at the bottom of the window to open the data region dialog. See Properties Window for further information on how to access commands.
4. On the data region dialog, on the **Groups** page, set the grouping expression. For example, group the data on StorePrice (=Fields!StorePrice.Value). See Group Data for further information.
5. On the same **Groups** page, set the Document map label to the value of the grouping expression. For example, =Fields!StorePrice.Value.
6. On the design surface, select a control inside the data region. For example, the TextBox in the detail row of the Table data region.
7. Right-click the control and select properties to open the Properties Window. In the command section of the Properties Window, click **Property dialog**.
8. In the dialog that appears, go to the **Navigation** page and under the Document map label, enter a text or an expression representing the control in the Document map.

Go to the preview tab or the Viewer to view the document map.



Note: In an Page report, you can also set grouping and set the document map label on the FixedPage dialog > Groups page. See Group Data to learn how to set groups on a fixed page.

To set the numbering style for Document Map levels

You can the set the numbering style for all document map levels using the **Report dialog** or in the Properties Window using the **DocumentMap** property.

1. In the Report Explorer select the **Report** item.
2. In the command section of the Properties Window, click Property dialog. This is a command to open the Report dialog. See Properties Window for more information on how to access commands.
3. In the Report dialog that appears, go to the Document Map page and set the **Source** property to one of the available value.
4. Set the **Numbering Style** to one of the available formats. The selected style would get applied to all the document levels and only to the selected source that you had set above.
5. Click **OK** to close the dialog.

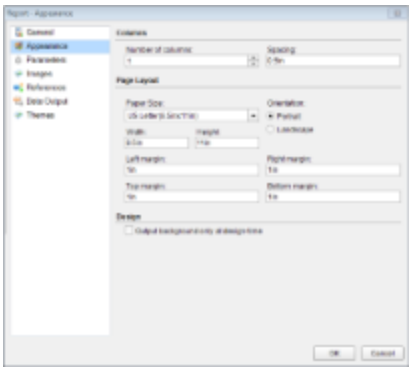
You can also set separate numbering style to different document map levels using the **DocumentMapLevelDesigner Collection Editor** dialog.

1. In the Report Explorer select the **Report** item.
2. Go to the Properties Window, click the **Levels (Collection)** property and then click the ellipsis button that appears.
3. In the **DocumentMapLevelDesigner Collection Editor** dialog that appears, consider the hierarchy of report bookmarks that appears in Document Map and add as many levels required using the **Add** button.
4. Select each level and set its **NumberingStyle** to any of the available format.
5. Click **OK** to close the dialog.

Note: Any customization made here gets directly applied to the Table of Contents control.

Change Page Size

You can change the size of your page through the **Appearance** page of the Report Dialog.



1. Click the gray area outside the design surface to select the report and under the Properties Window, click the Property dialog command.
2. In the Report dialog that appears, on the Appearance page, select from a list of pre-defined **Paper Size** options from the dropdown list. This automatically changes the Height and Width values of the page based on the selected size.
3. Set the **Orientation** of the page to Portrait or Landscape. This also modifies the Height and Width values of the page.

You can also set the page size through the **PageSize** property in the Properties Window.

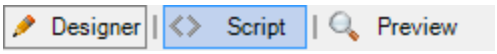
> **Note**: The unit of measure is based on your locale, but you can change it by typing in a different unit designator such as cm or pt after the number.

Add Code to Layouts Using Scripts

In a page report or a RDL report, you can use custom code in your expressions to extend the capabilities of your report. However, for complex functions, or functions you plan to use many times in the report, you also have the facility to embed the code within the report. You can also create and maintain a custom assembly for code that you want to use in multiple reports and refer to its methods in expressions.

Embed Code in the Report

Add a page report/RDL report template to your project and in the ActiveReports Designer that appears, add code like the following in the Script tab.



To call a function from the control's property

This is a simple example of a single method code block:

```
Public Function GetDueDate() as Date
    Return DateTime.Now.AddDays(30)
End Function
```

This is an expression used to call the method in the code block from the control's property. For example, you can call this function in the **Value** property of the Textbox control:

```
=Code.GetDueDate()
```

To use the custom constant and variable from the control's property

This is a simple example of how to define custom constants and variables in your code block:

```
Public Dim MyVersion As String = "123.456"
Public Dim MyDoubleVersion As Double = 123.456
Public Const MyConst As String = "444"
```

This is an expression to use the custom constant and variable in the code block from the control's property. For example, you can get the value of a variable or a constant in the **Value** property of the Textbox control:

```
=Code.MyVersion
=Code.MyDoubleVersion
=Code.MyConst
```

To call a global collection from the control's property

This is a simple example of a global collection code block where the code block references the report object to access the report parameter value:

```
Public Function ReturnParam() As String
    Return "param value = " + Report.Parameters!ReportParameter1.value.ToString()
End Function
```

This is an expression used to call a global collection in the code block from the control's property. For example, you can call the global collection in the **Value** property of the Textbox control:

```
=Code.ReturnParam()
```

Use instance-based Visual Basic .NET code in the form of a code block. You can include multiple methods in your code block, and access those methods from expressions in the control properties.

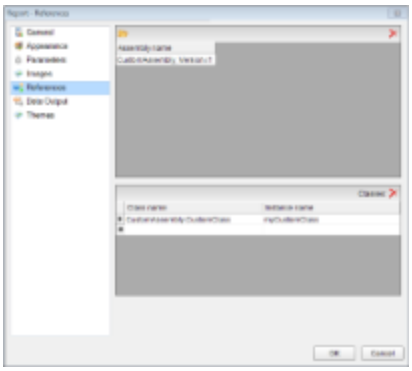
> **Note:** In a page report or a RDL report, you can use Visual Basic.NET as the script language. However, you can use both Visual Basic.Net and C# in your script for a section report.

Create custom assemblies

You can create custom assemblies in C# or Visual Basic .NET to make code available to multiple reports:

1. Create or find the assembly you wish to use. The assemblies can be found in the installed location: C:\Program Files (x86)\GrapeCity\ActiveReports 14\NuGet\{*Package name*}\lib\net462\{*Assembly*}
2. Make the assembly available to the report engine.
   - If you are embedding the designer or viewer controls in your own application, copy the assembly to the same location as your executable.
   - If you are using the included designer or viewer, copy the assembly into the ActiveReports assembly folder located at ...\GrapeCity\ActiveReports 14 by default.

     > **Note:** To make the assembly available for use in your own application and for use in designing reports for your application, copy it to both locations listed above. Alternatively, you can place the assembly in the Global Assembly Cache (C:\Windows\assembly).
3. Add an assembly reference to the report.
   - From the Report Menu, choose **Report Properties**.
   - In the Report dialog that appears, select the **References** and click the **Open** icon above the assembly name list to add your own assembly.
   - Go to the Class list below the assembly names and under **Class name**, enter the namespace and class name. Similarly, under **Instance name**, enter the name you want to use in your expressions.



4. Access the assembly through expressions.
    - To access static members (member denoted as `public static` in C# assemblies, or as `Public Shared` in Visual Basic assemblies):
      `=Namespace.Class.Member`
    - To access class instances:
      `=Code.InstanceName`

Freeze Rows and Columns (RDL Report)

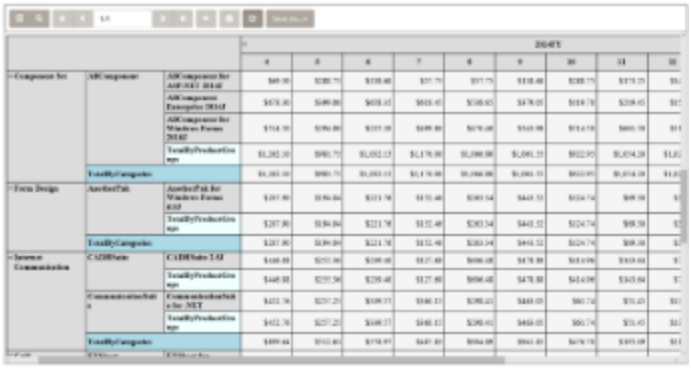
When you use a Table or a Tablix data region containing a large amount of data in an RDL report, the user must scroll to see all of the data. On scrolling the column or row headers out of sight, the data becomes difficult to understand.

> **Note:** The Frozen rows and columns feature is only available with RDL Reports.

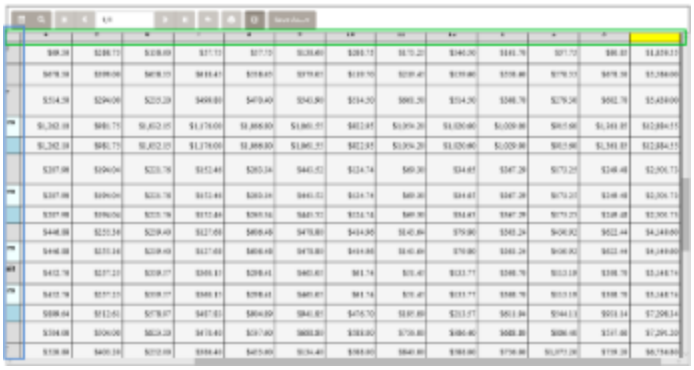
To alleviate this problem, we have added GrapeCity.ActiveReports.PageReportModel.Table.**FrozenRows** and GrapeCity.ActiveReports.PageReportModel.Table.**FrozenColumns** properties to the Table and Tablix data regions. The properties take effect in the JSViewer in Galley mode, and allow you to freeze headers so that they remain visible while scrolling through the data region. You can freeze as many rows or columns as you have headers in the data region.

- If your data stretches downward, set the FrozenRows property to a value to float the column headers when scrolling.
- If your data stretches to the right, set the FrozenColumns property to a value to float the row headers when scrolling.
- If your data stretches both downward and to the right, set both FrozenRows and FrozenColumns properties.

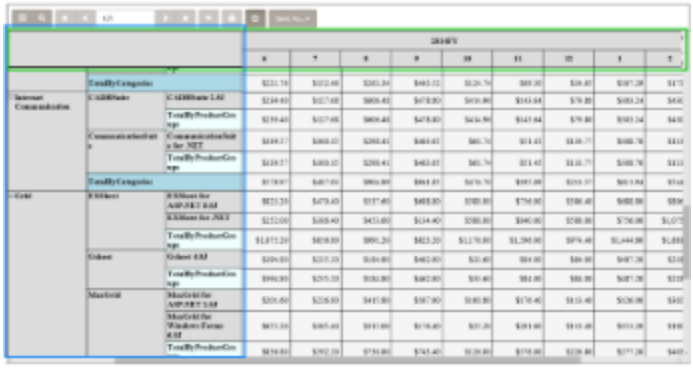
Here is an RDL report with a Tablix data region displayed in the JSViewer in Galley mode.



When you scroll to view more rows and columns of data, the row and column headers scroll out of view, like this.



When you set FrozenColumns = 3 and FrozenRows = 2, the three row headers and two column headers float when the user scrolls through the data, like this.



If any header cells that you want to freeze are merged, you should not set the FrozenRows or FrozenColumns property to a value that would split a merged cell. For example, in the image above, there is an empty merged cell at the top left corner. This cell prevents you from setting FrozenRows to a value less than 2, because it would split the merged cell. The same cell also prevents you from setting FrozenColumns to a value less than 3, because that would also split the merged cell.

Add Page Numbers

You can choose the page numbering format for your Page reports/RDL reports by selecting from a list of pre-defined formats or by creating a custom page numbering expression.

Adding page numbers to a report

There are two ways to add page numbering to a report.

- From the Report Explorer, under the **Common Values** node, drag a pre-defined page numbering format and drop it directly onto the report design surface.
- Or add a TextBox control to the report design surface and in the **Value** property of the control, use the Expression Editor Dialog to select a page numbering expression from the **Common Values** node.

Usually a page number is added to a report header or footer, but you can add it anywhere on the layout page.

Pre-defined page numbering formats

You can find the pre-defined page numbering formats listed in the Report Explorer under the **Common Values** node, and in the Expression Editor under the Common Values field.

Predefined Format Descriptions

Numbering Format	Description
Page N of M	This format displays the current page out of the total number of pages in the report. Here **N** signifies the current page of a report and **M** the total number of report pages. Use the following expression to set this page numbering format: `="Page " & Globals!PageNumber & " of " & Globals!TotalPages`
Page N of M (Section)	This format displays the current page out of the total number of pages of a grouped report section. Here **N** signifies the current page of a grouped report section and **M** signifies the total number of pages in a grouped report section. Use the following expression to set this page numbering format: `="Page " & Globals!PageNumberInSection & " of " & Globals!TotalPagesInSection`
Page N of M (Cumulative)	This format displays the current page out of the total number of cumulative pages in a report. Here **N** signifies the current page of the report and **M** signifies the total number of cumulative pages in a report. Use the following expression to set this page numbering format: `="Page " & Globals!CumulativePageNumber & " of " & Globals!CumulativeTotalPages`
Page Number	This format displays only the current page number of a report. Use the following expression to set this page numbering format: `=Globals!PageNumber`
Page Number (Section)	This format displays only the current page number of a specific grouped report section. Use the following expression to set this page numbering format: `=Globals!PageNumberInSection`
Total Pages	This format displays only the total number of pages in the report. Use the following expression to set this page numbering format: `=Globals!TotalPages`
Total Pages (Section)	This format displays only the total number of pages in specific grouped report section. Use the following expression to set this page numbering format: `=Globals!TotalPagesInSection`
Cumulative Page Number	This format displays only the current cumulative page number of the report. Use the following expression to set this page numbering format: `=Globals!CumulativePageNumber`
Cumulative Total Pages	This format displays only the total number of cumulative pages in a report. Use the following expression to set this page numbering format: `=Globals!CumulativeTotalPages`

Tip: In addition to modifying the page numbering expression in the Expression Editor, you can also modify the pre-defined formats directly in the control on the design surface.

Custom page numbering formats

Use the following steps to create your own page numbering format.

To create a custom page numbering format

1. From the toolbox, drag and drop a Textbox control onto the report design surface.
2. With the Textbox selected on the report, under the Properties window, click the Property dialog link. This is a command to open the TextBox dialog. See Properties Window for more on how to access commands.
3. In the **TextBox - General** dialog that appears, in the **Value** field, enter a page numbering expression like the following: `=Globals!PageNumber & "/" & Globals!TotalPages`
4. Click **OK** to close the dialog.
5. Select the Preview tab. Page numbers appear in the expression format you set in the **Value** field above, in this case, for a one-page report, "1/1."

Add Page Breaks in RDL (RDL Report)

In a page layout, you can add page breaks in a RDL report, using the **PageBreakAtStart** and **PageBreakAtEnd** properties of the report control.

You can set a page break before or after the Container control. It is also possible to force a page break before or after the following data regions or their groups:

- List
- Table
- Tablix
- Chart

Use the following steps to set page breaks in the report from the control dialogs:

To add a page break before or after a report control

1. On the design surface, select the report control on which you want to add a page break and in the command section of the Properties Window, click **Property dialog**. This is a command to open the control's dialog. See Properties Window for more information on how to access commands.
2. In the control's dialog that appears, on the General page, under Page Breaks, select the check box for **Insert a page break before this control** or **Insert a page break after this control** or both.
3. Click the **OK** button to save the changes and to close the dialog.
4. Go to the preview tab to view the result.

To add a page break before or after a group

1. On the design surface, select the report control containing a group and in the command section of the Properties Window, click **Property dialog**. This is a command to open the control's dialog. See Properties Window for more information on how to access commands.
2. In the control's dialog that appears, go to the **Groups** or **Detail Grouping** page whichever is available.
3. On the Groups or Detail Grouping page, go the **Layout** tab and select the check box for **Page break at start** or **Page break at end** or both.
4. Click the **OK** button to save the changes and to close the dialog.
5. Go to the preview tab to view the result.

Section Report How To

Learn to perform common tasks in Section reports with quick how-to topics.

Report Data

See step-by-step instructions for performing common tasks using ActiveReports.

In this section

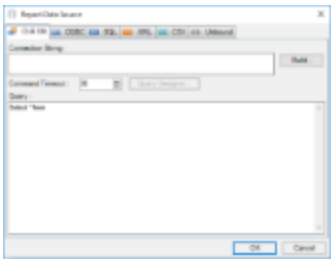
Bind Reports to a Data Source

At design time, you can connect a section report to a data source through the **Report Data Source** dialog. You can access the Report Data Source dialog by doing one of the following:

- On the detail section band, click the Data Source Icon.

- Click the gray area around the design surface and in the commands section at the bottom of the Properties Window, click the **Edit Data Source** command.

There are four tabs in the dialog for the four most commonly used data sources.



The following steps take you through the process of binding reports to each data source. These steps assume that you have already added an ActiveReports 14 Section Report template in a Visual Studio project. See Quick Start further

information on adding different report layouts.

To use the OLE DB data source

1. In the Report Data Source dialog, on the **OLE DB** tab, click the Build button next to Connection String.
2. In the Data Link Properties window that appears, select **Microsoft Jet 4.0 OLE DB Provider** and click the **Next** button to move to the Connection tab.
3. Click the ellipsis (...) button to browse to your database, for example the NWind.mdb sample database. Click **Open** once you have selected the appropriate database path.
4. Click the **Test Connection** button to see if you have successfully connected to the database.
5. Click **OK** to close the Data Link Properties window and return to the Report Data Source dialog. Notice that the Connection String field gets filled automatically.
6. In the **Query** field on the **OLE DB** tab, enter a SQL query to select the data that you want use from the connected database. For example, `Select * From CUSTOMERS`
   OR
   In the Report Data Source dialog, click on Query Designer button to access Visual Query Designer for creating SQL queries. See Visual Query Designer for further information on how to create a query using the interactive query designer.
7. Click **OK** to save the data source and return to the report design surface.

To use the ODBC data source

Before you connect to a ODBC data source, you must install a ODBC driver and set up a ODBC data source. For more information, see How To: Setup an ODBC Data Source.

1. In the Report Data Source dialog, click the **ODBC** tab.
2. In the **Connection String** field on the **ODBC** tab, enter a connection string to connect to the database. For example, `Provider=MSDASQL;Persist Security Info=False;DSN=MS Access Database`
3. In the **Query** field on the **ODBC** tab, enter a SQL query to select the data that you want use from the connected database. For example, `Select * From CUSTOMERS`
4. Click **OK** to save the data source and return to the report design surface.

To use the SQL data source

1. In the Report Data Source dialog, on the **SQL** tab, click the Build button next to Connection String.
2. In the Data Link Properties window that appears, select **Microsoft OLE DB Provider for SQL Server** and click the **Next** button to move to the Connection tab.
3. On the Connection tab of the Data Link Properties window:
   - In the **Select or enter server name** field, select your server from the drop down list.
   - Under **Enter information to log on to the server**, select the Windows NT security credentials or your specific user name and password.
   - Under **Select the database on the server**, select a database from the server or attach a database file.
   - Click the **Test Connection** button to see if you have successfully connected to the database.
4. Click **OK** to close the Data Link Properties window and to return to the Report Data Source dialog. Notice that the Connection String field gets filled automatically.
5. In the **Query** field on the **SQL** tab, enter a SQL query to select the data that you want use from the connected database. For example, `Select * From CUSTOMERS`
   OR
   In the Report Data Source dialog, click on Query Designer button to access Visual Query Designer for creating SQL queries. See Visual Query Designer for further information on how to create a query using the interactive query

designer.
6. Click **OK** to save the data source and return to the report design surface.

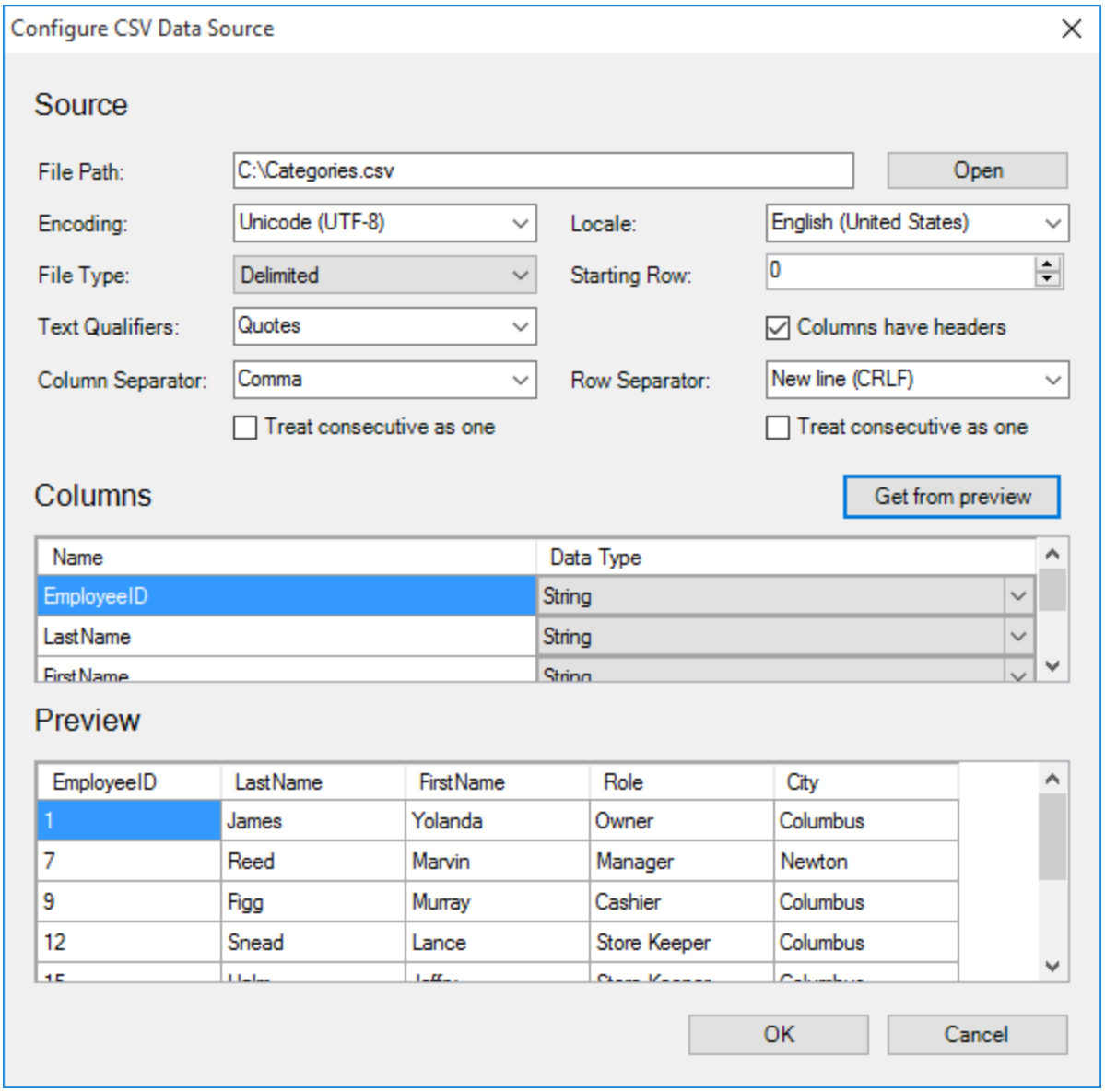
To use the XML data source

1. In the Report Data Source dialog, on the **XML** tab, click the ellipsis (...) button next to File URL field.
2. In the **Open File** window that appears, navigate to your XML data file to select it and click the **Open** button. You can use a sample XML data file (Customer.xml) available on GitHub.
3. In the **Recordset Pattern** field, enter a valid XPath expression like: `//CUSTOMER`
4. Click **OK** to save the data source and return to the report design surface.

You also have the option to use an unbound or an IEnumerable data source. See the following procedures to implement these data source connections in code.

To use the CSV data source

1. In the Report Data Source dialog, on the **CSV** tab, click the **Build** button next to Connection String.
2. Specify the **File Path** by clicking the **Open** button and selecting the .csv file. You can also enter a relative path to the csv file here.
3. Set the options in the wizard. See the **Sample CSV Connection String** drop-down under Report Data Source Dialog for further details.
4. To edit the Name, Width (if applicable), and Data Type of columns shown in the Preview, click the **Get from preview** button. Note that Width is applicable only for Fixed data type.



5. Click **OK** to save the changes and close the dialog. The **Connection String** tab displays the generated connection string. You can validate the connection string by clicking the **Validate DataSource** icon .
6. Click **OK** on the lower right corner to close the dialog. You have successfully connected the report to a CSV data source. Note that the dataset for the CSV data source is added automatically.

To use an Unbound data source

To create a data connection

1. Add an Imports (VisualBasic.NET) or using (C#) statement for System.Data and System.Data.Oledb namespaces.
2. Right-click the gray area outside the design surface to select the report and select Properties.
3. In the Properties window that appears, click the Events icon to view the available events for the report.
4. In the events list, double-click the **ReportStart** event**.** This creates an event-handling method for the ReportStart event in code.
5. Add the following code to the handler.

To write code in VisualBasic.NET

Visual Basic.NET code. Paste above the ReportStart event.

```
Dim m_cnnString As String
Dim sqlString As String
Dim m_reader As OleDbDataReader
Dim m_cnn As OleDbConnection
```

Visual Basic.NET code. Paste inside the ReportStart event.

```
'Set data source connection string.
m_cnnString = "Provider=Microsoft.Jet.OLEDB.4.0;" _
        + "Data Source=[User Folder]\Samples14\Data\NWIND.mdb;Persist Security
Info=False"
'Set data source SQL query.
sqlString = "SELECT * FROM categories INNER JOIN products ON categories.categoryid
" _
        + "= products.categoryid ORDER BY products.categoryid, products.productid"
'Open connection and create DataReader.
m_cnn = New OleDb.OleDbConnection(m_cnnString)
Dim m_Cmd As New OleDb.OleDbCommand(sqlString, m_cnn)
If m_cnn.State = ConnectionState.Closed Then
   m_cnn.Open()
End If
m_reader = m_Cmd.ExecuteReader()
```

To write code in C#

C# code. Paste above the ReportStart event.

```
private static OleDbConnection m_cnn;
private static OleDbDataReader m_reader;
private string sqlString;
private string m_cnnString;
```

C# code. Paste inside the ReportStart event.

```
//Set data source connection string.
m_cnnString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source="
        + @"=[User Folder]\Samples14\Data\NWIND.mdb;Persist Security Info=False";
//Set data source SQL query.
sqlString = "SELECT * FROM categories INNER JOIN products"
        + " ON categories.categoryid = products.categoryid"
        + " ORDER BY products.categoryid, products.productid";
//Open connection and create DataReader.
m_cnn = new OleDbConnection(m_cnnString);
OleDbCommand m_Cmd = new OleDbCommand(sqlString,m_cnn);
if(m_cnn.State == ConnectionState.Closed)
{
  m_cnn.Open();
```

```
}
m_reader = m_Cmd.ExecuteReader();
```

To close the data connection

1. Right-click the gray area outside the design surface to select the report and select Properties.
2. In the Properties window that appears, click the Events icon to view the available events for the report.
3. In the events list, double-click the **ReportEnd** event. This creates an event-handling method for the ReportEnd event.
4. Add the following code to the handler.

To write the code in Visual Basic

Visual Basic.NET code. Paste inside the ReportEnd event.

```
m_reader.Close()
m_cnn.Close()
```

To write the code in C#

C# code. Paste inside the ReportEnd event.

```
m_reader.Close();
m_cnn.Close();
```

To create a fields collection

1. Right-click the gray area around the design surface to select the report and select Properties.
2. In the Properties window that appears, click the Events icon to view the available events for the report.
3. In the events list, double-click **DataInitialize** event. This creates an event-handling method for the report's DataInitialize event.
4. Add code to the handler to add fields to the report's fields collection.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste inside the DataInitialize event.

```
Fields.Add("CategoryName")
Fields.Add("ProductName")
Fields.Add("UnitsInStock")
Fields.Add("Description")
```

To write the code in C#

C# code. Paste inside the DataInitialize event.

```
Fields.Add("CategoryName");
Fields.Add("ProductName");
Fields.Add("UnitsInStock");
Fields.Add("Description");
```

To populate the fields

1. Right-click the gray area around the design surface to select the report and select Properties.
2. In the Properties window that appears, click the Events icon to view the available events for the report.
3. In the events list, double-click the **FetchData** event**.** This creates an event-handling method for the report's FetchData event.
4. Add the following code to the handler to retrieve information to populate the report fields.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste inside the FetchData event.

```
Try
    m_reader.Read()
    Me.Fields("CategoryName").Value = m_reader("CategoryName")
    Me.Fields("ProductName").Value = m_reader("ProductName")
    Me.Fields("UnitsInStock").Value = m_reader("UnitsInStock")
    Me.Fields("Description").Value = m_reader("Description")
    eArgs.EOF = False
Catch ex As Exception
    eArgs.EOF = True
End Try
```

To write the code in C#

C# code. Paste inside the FetchData event.

```
try
{
    m_reader.Read();
    Fields["CategoryName"].Value = m_reader["CategoryName"].ToString();
    Fields["ProductName"].Value = m_reader["ProductName"].ToString();
    Fields["UnitsInStock"].Value = m_reader["UnitsInStock"].ToString();
    Fields["Description"].Value = m_reader["Description"].ToString();
    eArgs.EOF = false;
}
catch
{
    eArgs.EOF = true;
}
```

Tip: In order to view the added data at run time, add controls to your report and assign their DataField property to the name of the fields you added in code while creating a field collection.

Caution: Do not access the Fields collection outside the DataInitialize and FetchData events. Accessing the Fields collection outside of these events is not supported, and has unpredictable results.

To use the IEnumerable data source

1. Right-click the design surface and select View Code.
2. Add the following code inside the class declaration of the report:

To create a data source in Visual Basic

Visual Basic.NET code. Paste inside the class declaration of the report.

```
Private datasource1 As IEnumerator(Of String) = Nothing
Dim list As List(Of String)= Nothing
```

Visual Basic.NET code. Paste inside the class declaration of the report.

```
Private Function GetIEnumerableData() As IEnumerable(Of String)
    For i As Integer = 1 To 10
       list.Add(String.Format("TestData_{0}", i.ToString()))
    Next
    Return list
End Function
```

To create a data source in C#

C# code. Paste inside the class declaration of the report.

```
private IEnumerator<string> datasource = null;
```

C# code. Paste inside the class declaration of the report.

```
private IEnumerable<string> GetIEnumerableData()
{
    for (int i = 1; i <= 10; i++)
    {
        yield return string.Format("TestData_{0}", i.ToString());
    }
}
```

3. On the design surface, right-click the gray area around the design surface to select the report and select Properties.
4. In the Properties window that appears, click the Events icon to view the available events for the report.
5. Double-click the **DataInitialize** event. This creates an event-handling method for the report's DataInitialize event.
6. Add the following code to the handler to add fields to the report's Fields collection.

To add fields in Visual Basic

Visual Basic.NET code. Paste inside the DataInitialize event.

```
Me.Fields.Add("TestField")
Me.list = New List(Of String)
datasource1 = GetIEnumerableData().GetEnumerator()
```

To add fields in C#

C# code. Paste inside the DataInitialize event.

```
this.Fields.Add("TestField");
datasource = GetIEnumerableData().GetEnumerator();
```

7. Repeat steps 3 and 4 to open the events list in the property window.
8. Double-click the **FetchData** event. This creates an event-handling method for the report's FetchData event.

9. Add code to the handler to retrieve information to populate the report fields.

To populate fields in Visual Basic

Visual Basic.NET code. Paste inside the FetchData event.

```
If datasource1.MoveNext() Then
      Me.Fields("TestField").Value = datasource1.Current
      eArgs.EOF = False
Else
      eArgs.EOF = True
End If
```

To populate fields in C#

C# code. Paste inside the FetchData event.

```
if (datasource.MoveNext())
{
    this.Fields["TestField"].Value = datasource.Current;
    eArgs.EOF = false;
}
else
    eArgs.EOF = true;
```

Tip: In order to view the added data at run time, add controls to your report and assign their DataField property to the name of the fields you added in code while creating a field collection.

Modify Data Sources at Run Time

In a section report, you can modify your data source at run time. Follow the steps below to connect your report to the NWind.mdb sample database at run time.

To find the database path

Note: These steps assume that the data path is defined in the following registry key.

1. Right-click the design surface, and select **View Code** to display the code view for the report.
2. Add the following code to the report to access the sample database path from the registry.

To write the code in Visual Basic

The following example shows what the code for the function looks like.

Visual Basic.NET code. Paste below the Imports GrapeCity.ActiveReports statement at the top of the code view.

```
Imports System
Imports Microsoft.Win32
```

This creates a function for getDatabasePath.

Visual Basic.NET code. Paste inside the getDatabasePath function.

```
Private Function getDatabasePath() As String
Dim regKey As RegistryKey
regKey = Registry.LocalMachine
regKey = regKey.CreateSubKey("SOFTWARE\\GrapeCity\\ActiveReports\\v14")
getDatabasePath = CType(regKey.GetValue(""), String)
End Function
```

To write the code in C#

The following example shows what the code for the function looks like.

C# code. Paste below the using GrapCity.ActiveReports statement at the top of the code view.

```
using Microsoft.Win32;
using System;
```

This creates a function for getDatabasePath.

C# code. Paste BELOW the getDatabasePath function.

```
private string getDatabasePath()
{
RegistryKey regKey = Registry.LocalMachine;
regKey = regKey.CreateSubKey("SOFTWARE\\GrapeCity\\ActiveReports\\v14");
return ((string)(regKey.GetValue("")));
}
```

To change the data source at run time

1. Double-click the gray area outside the design surface to create an event-handling method for the **ReportStart** event.
2. Add the following code to the handler to change the data source at run time.

To write the code in Visual Basic.NET

The following example shows what the code for the method looks like.

Visual Basic.NET code. Paste above the ReportStart event.

```
Dim conn As System.Data.OleDb.OleDbConnection
Dim reader As System.Data.OleDb.OleDbDataReader
```

Visual Basic.NET code. Paste inside the ReportStart event.

```
Dim dbPath As String = getDatabasePath()
Dim connString As String = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" + dbPath
+ "\NWIND.mdb"
conn = New System.Data.OleDb.OleDbConnection(connString)
```

```
Dim cmd As New System.Data.OleDb.OleDbCommand("SELECT * FROM Products WHERE
UnitPrice = 18", conn)
conn.Open()
reader = cmd.ExecuteReader()
Me.DataSource = reader
```

To write the code in C#

The following example shows what the code for the method looks like.

C# code. Paste above the ReportStart event.

```
private static System.Data.OleDb.OleDbConnection conn;
private static System.Data.OleDb.OleDbDataReader reader;
```

C# code. Paste inside the ReportStart event.

```
string dbPath = getDatabasePath();
string connString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" + dbPath +
"\\NWIND.mdb";
conn = new System.Data.OleDb.OleDbConnection(connString);
System.Data.OleDb.OleDbCommand cmd = new System.Data.OleDb.OleDbCommand("SELECT *
FROM Products WHERE UnitPrice = 18", conn);
conn.Open();
reader = cmd.ExecuteReader();
this.DataSource = reader;
```

To close the data connection

1. Right-click the gray area outside the design surface and select Properties.
2. In the Properties Window that appears, click the Events button. A list of report events appear.
3. Select the ReportEnd event and double click to create an event-handling method.
4. Add the following code to the handler to close the data connection.

To write the code in Visual Basic

The following example shows what the code for the method looks like.

Visual Basic.NET code. Paste inside the ReportEnd event.

```
reader.Close()
conn.Close()
```

To write the code in C#

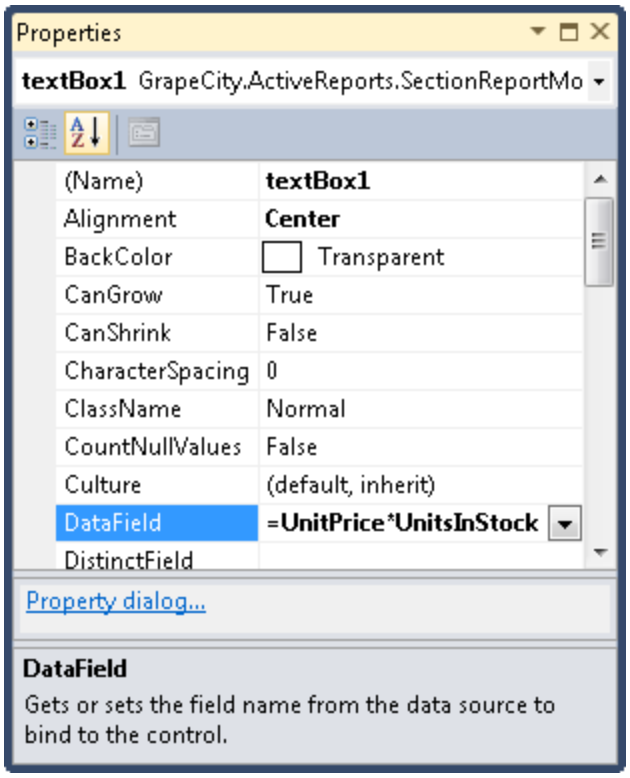
The following example shows what the code for the method looks like.

C# code. Paste inside the ReportEnd event.

```
reader.Close();
conn.Close();
```

Add Field Expressions

In a section report, expressions can be used in the **DataField** property to specify textbox output in a report, such as date/time, mathematical calculations or conditional values. All field expressions used in the DataField property begin with the equals sign (=).



To use a mathematical expression

Set the DataField property of a textbox control to a mathematical calculation.

Example:

```
=UnitPrice+5
```

```
=Quantity-5
```

```
=Quantity*UnitPrice
```

```
=UnitPrice/QuantityPerUnit
```

To use a substring

Set the DataField property of a textbox control to the required substring. While setting up grouping, change the GroupHeader's DataField property to the same substring.

Example:

```
=ProductName.Substring(0, 1)
```

To use date/time

Set the DataField property of a textbox control similar to the following expression to display date/time values.

Example:

```
=System.DateTime.Now.ToString()
```

To create a conditional value

Set the DataField property of a textbox control to the conditional statement as desired.

Example:

```
=(UnitsInStock > 0)?"In Stock":"Backorder"
```

To concatenate fields

Set the DataField property of a textbox control similar to the following expression to display concatenated fields.

Example:

```
="There are " + UnitsInStock + " units of " + ProductName + " in stock."
```

```
=TitleOfCourtesy + " " + FirstName + " " + LastName
```

Note: ActiveReports automatically handles null values, replacing them with an empty string.

To round a calculation

Set the DataField Property of a textbox control like the following example.

Example:

```
=(double)System.Math.Round(UnitsInStock/10)
```

To use modular division

Set the DataField property of a textbox control like the following example to get the remainder (2 in this case).

Example: `=22%(5)`

To replace a null value

Set the DataField property of a textbox control like the following example to replace null with your own value.

Example:

```
=(UnitsInStock == System.DBNull.Value) ? "No Units In Stock" : UnitsInStock
```

Display Page Numbers and Report Dates

With the ReportInfo control available in a section report, you can display page numbers and report dates and times by selecting a value in the **FormatString** property. This property provides the following pre-defined options for page

numbering and date and time formatting.

Predefined Options and their Description

Numbering Format	Description
Page {PageNumber} of {PageCount} on {RunDateTime}	Display the page numbers along with Date and Time in the following format : `Page 1 of 100 on 1/31/2012 2:45:50 PM`
Page {PageNumber} of {PageCount}	Display the only the page numbers in the following format : `Page 1 of 100`
{RunDateTime:}	Display the Date and Time in the following format : `1/31/2012 2:45:50 PM`
{RunDateTime: M/d}	Display the Date in the following format : `1/31`
{RunDateTime: M/d/yy}	Display the Date in the following format : `1/31/12`
{RunDateTime: M/d/yyyy}	Display the Date in the following format : `1/31/2012`
{RunDateTime: MM/dd/yy}	Display the Date in the following format : `01/31/12`
{RunDateTime: MM/dd/yyyy}	Display the Date in the following format : `01/31/2012`
{RunDateTime: d-MMM}	Display the Date in the following format : `31-Jan`
{RunDateTime: d-MMM-yy}	Display the Date in the following format : `31-Jan-12`
{RunDateTime: d-MMM-yyyy}	Display the Date in the following format : `31-Jan-2012`
{RunDateTime: dd-MMM-yy}	Display the Date in the following format : `31-Jan-12`
{RunDateTime: dd-MMM-yyyy}	Display the Date in the following format : `31-Jan-2012`
{RunDateTime: MMM-yy}	Display the Date in the following format : `Jan-12`
{RunDateTime: MMM-yyyy}	Display the Date in the following format : `Jan-2012`
{RunDateTime: MMMM-yy}	Display the Date in the following format : `January-12`
{RunDateTime: MMMM-yyyy}	Display the Date in the following format : `January-2012`
{RunDateTime: MMMM d,yyyy}	Display the Date in the following format : `January 31, 2012`
{RunDateTime: M/d/yy h:mm tt}	Display the Date and Time in the following format : `1/31/12 2:45 PM`
{RunDateTime: M/d/yyyy h:mm tt}	Display the Date and Time in the following format : `1/31/2012 2:45 PM`
{RunDateTime: M/d/yy h:mm}	Display the Date and Time in the following format : `1/31/12 2:45`

{RunDateTime: M/d/yyyy h:mm}	Display the Date and Time in the following format : `1/31/2012 2:45`

Page numbering can also be set to a group level using the **SummaryGroup** and **SummaryRunning** properties.

These steps assume that you have already added a Section Report to a project in Visual Studio. See Quick Start for more information.

To display page numbers and report dates on a report

1. From the ActiveReports 14 Section Report tab in the toolbox, drag the **ReportInfo** control to the desired location on the design surface.
2. With the ReportInfo control selected in the Properties Window, drop down the **FormatString** property.
3. Select the pre-defined format that best suits your needs.

> **Tip:** You can customize the pre-defined formats in the Properties Window. For example, if you change the **FormatString** property to **Page {PageNumber} / {PageCount}**, it shows the first page number as **Page 1/1**. For more information on creating formatting strings, see the Date, Time, and Number Formatting topic.

To display page numbers and page count at the group level

1. From the ActiveReports 14 Section Report tab in the toolbox, drag the ReportInfo control to the **GroupHeader** or **GroupFooter** section of the report and set the FormatString property as above.
2. With the ReportInfo control still selected in the Properties Window, drop down the **SummaryGroup** property and select the group for which you want to display a page count.
3. Drop down the **SummaryRunning** property and select **Group**.

Load a File into a RichTextBox Control

In a section layout, you can load an RTF or an HTML file into the RichTextBox control both at design time and at run time . Following is a step-by-step process that helps you load these files into the RichTextBox control.

These steps assume that you have already added a Section Report (code based) template in a Visual Studio project and have placed a RichTextBox control inside its detail section. See Quick Start for more information.

> **Caution**: Do not attempt to load a file into a RichTextBox in a section that repeats. After the first iteration of the section, the RTF or HTML file is already in use by that first iteration and returns "file in use" errors when that section is processed again.

To write an RTF file to load into a RichTextBox control

1. Open wordpad, and paste the following formatted text into it.

Paste the following section into an RTF File.

Customer List by Country

Argentina

- Rancho grande
- Océano Atlántico Ltda.
- Cactus Comidas para llevar

Austria

- Piccolo und mehr
- Ernst Handel

Belgium

- Suprêmes délices
- Maison Dewey

Brazil

- Familia Arquibaldo
- Wellington Improtadora
- Que Delícia
- Tradição Hipermercados
- Ricardo Adocicados
- Hanari Carnes
- Queen Cozinha
- Comércio Mineiro
- Gourmet Lanchonetes

2. Save the file as **sample.rtf** in the debug directory inside the bin folder of your project.

> **Note**: The RichTextBox control is limited in its support for advanced RTF features such as the ones supported by Microsoft Word. In general, the features supported by WordPad are supported in this control.

To load an RTF file into the RichTextBox control at design time

1. On the report design surface, add a RichTextBox control.
2. With the RichTextBox control selected, at the botton of the Properties Window, click the **Load File** command. See Properties Window for a description of commands.
3. In the Open dialog that appears, browse to an *.RTF file (For example, sample.rtf) and click the Open button to load the file in the RichTextBox control.

To load an RTF file into the RichTextBox control at run time

> **Note**: The RichTextBox control has limited support for advanced RTF features such as the ones supported by Microsoft Word. Therefore, use a WordPad for obtaining best results.

These steps assume that the RTF file (for example, sample.rtf) to load has been saved in the bin/debug directory of your project.

1. Right-click the report and select View Code to open the code view.
2. Add an Imports (Visual Basic.NET) or using (C#) statement at the top of the code view for the GrapeCity.ActiveReports.SectionReportModel namespace.

3. In the design view, double-click the detail section of the report to create an event-handling method for the Detail Format event.
4. Add the following code to the handler to load the RTF file into the RichTextBox control.

   The following example shows what the code for the method looks like.

   To write the code in Visual Basic.NET

   Visual Basic.NET code. Paste INSIDE the Detail1_Format event.

```
Dim streamRTF As New
System.IO.FileStream(System.Windows.Forms.Application.StartupPath + "\sample.rtf",
System.IO.FileMode.Open)
Me.RichTextBox1.Load(streamRTF, RichTextType.Rtf)
```

   To write the code in C#

   C# code. Paste INSIDE the detail_Format event.

```
System.IO.FileStream streamRTF = new
System.IO.FileStream(System.Windows.Forms.Application.StartupPath + "\\sample.rtf",
System.IO.FileMode.Open);
this.richTextBox1.Load(streamRTF, RichTextType.Rtf);
```

   Note: The Application.Startup path code does not work in preview mode. You must run the project in order to see the file load.

To write an HTML file to load into a RichTextBox control

1. Open a Notepad, and paste the following HTML code into it.

   HTML code. Paste in a NotePad file.

```
<html>
<body>
<center><h1>Customer List by Country</h1></center>
<h1>Argentina</h1>
<ul>
<li>Rancho grande
<li>Océano Atlántico Ltda.
<li>Cactus Comidas para llevar
</ul>
<h1>Austria</h1>
<ul>
<li>Piccolo und mehr
<li>Ernst Handel
</ul>
<h1>Belgium</h1>
<ul>
<li>Suprêmes délices
<li>Maison Dewey
```

```
</ul>
<h1>Brazil</h1>
<ul>
<li>Familia Arquibaldo
<li>Wellington Improtadora
<li>Que Delícia
<li>Tradição Hipermercados
<li>Ricardo Adocicados
<li>Hanari Carnes
<li>Queen Cozinha
<li>Comércio Mineiro
<li>Gourmet Lanchonetes
</ul>
</body>
</html>
```

2. Save the file as **sample.html**.

To load an HTML file into the RichTextBox control at design time

1. On the report design surface, add a RichTextBox control.
2. With the RichTextBox control selected, at the bottom of the Properties Window, click the Load File command. See Properties Window for a description of commands.
3. In the Open dialog that appears, browse to an *.html file (For example, sample.html) and click the Open button to load the file in the RichTextBox control.

To load an HTML file into a RichTextBox control at run time

These steps assume that the HTML file (for example, sample.html) to load has been saved in the bin/debug directory of your project.

1. Right-click the report and select View Code to open the code view.
2. Add an Imports (Visual Basic.NET) or using (C#) statement at the top of the code view for the GrapeCity.ActiveReports.SectionReportModel namespace.
3. In the design view, double-click the detail section of the report to create an event-handling method for the Detail Format event.
4. Add code to the handler to load the HTML file into the RichText control.

   The following example shows what the code for the method looks like.

   To write the code in Visual Basic.NET

   Visual Basic.NET code. Paste INSIDE the Detail1_Format event.

   ```
   Dim streamHTML As New
   System.IO.FileStream(System.Windows.Forms.Application.StartupPath + "\sample.HTML",
   System.IO.FileMode.Open)
   Me.RichTextBox1.Load(streamHTML, RichTextType.Html)
   ```

   To write the code in C#

C# code. Paste INSIDE the detail_Format event.

```
System.IO.FileStream streamHTML = new
System.IO.FileStream(System.Windows.Forms.Application.StartupPath +
"\\sample.html", System.IO.FileMode.Open);
this.richTextBox1.Load(streamHTML, RichTextType.Html);
```

Note: The Application.Startup path code does not work in preview mode. You must run the project in order to see the file load.

Use Custom Controls on Reports

In a section report, ActiveReports allows you to drop a third party control onto the report design surface where it is recognized as a custom control. You can access its properties using type casting.

In the following steps, we use hidden textbox controls to populate a Visual Studio TreeView control. These steps assume that you have already added a Section Report (code-based) template in a Visual Studio project. See Quick Start for more information.

To add the TreeView control to a report

1. From the Visual Studio toolbox **Common Controls** tab, drag and drop a **TreeView** control onto the detail section of a report.
2. Notice that in the Properties window, the control is called **CustomControl1**.

To add data and hidden TextBox controls to the report

1. Connect the report to the sample Nwind.mdb. The following steps use the Orders table from the NWind database. The NWIND.mdb file can be downloaded from GitHub: ..\Samples14\Data\NWIND.mdb.
2. From the Report Explorer, drag and drop the following fields onto the detail section of the report:
   - ShipCountry
   - ShipCity
   - CustomerID
   - EmployeeID
3. On the design surface, select all four TextBox controls, and in the Properties window, change their **Visible** property to **False**.

To create a function to add nodes to the TreeView control

1. Right-click the design surface and select **View Code** to see the code view for the report.
2. Add the following code inside the report class to add a function to the report for adding nodes to the TreeView control.

   The following examples show what the code for the function looks like.

   To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the report class.

```
Private Function AddNodeToTreeView(ByVal colNodes As TreeNodeCollection, ByVal
sText As String) As TreeNode
```

```
    Dim objTreeNode As TreeNode
    objTreeNode = New TreeNode(sText)
    colNodes.Add(objTreeNode)
    Return objTreeNode
End Function
```

To write the code in C#

C# code. Paste INSIDE the report class.

```
private TreeNode AddNodeToTreeView(TreeNodeCollection colNodes, string sText)
{
    TreeNode objTreeNode;
    objTreeNode = new TreeNode(sText);
    colNodes.Add(objTreeNode);
    return objTreeNode;
}
```

To access the TreeView control properties in code and assign data

1. On the report design surface, double-click the detail section to create an event-handling method for the **Detail_Format** event.
2. Add the following code to the handler to access the **TreeView** properties and assign data from the hidden TextBox controls.

   The following example shows what the code for the method looks like.

   To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the Detail Format event.

```
'Type cast the custom control as a TreeView
Dim TreeView1 As New TreeView
TreeView1 = CType(Me.CustomControl1.Control, TreeView)

'Create a tree node
Dim objCountryTreeNode As TreeNode
'Assign the text from a hidden textbox to the node
objCountryTreeNode = AddNodeToTreeView(TreeView1.Nodes, Me.txtShipCountry1.Text)
'Add a second-level node
AddNodeToTreeView(objCountryTreeNode.Nodes, Me.txtShipCity1.Text)
'Expand the top-level node so the second-level node is in view
objCountryTreeNode.Expand()

'Create a second top-level node
Dim objCustomerTreeNode As TreeNode
objCustomerTreeNode = AddNodeToTreeView(TreeView1.Nodes, Me.txtCustomerID1.Text)
AddNodeToTreeView(objCustomerTreeNode.Nodes, Me.txtEmployeeID1.Text)
objCustomerTreeNode.Expand()
```

To write the code in C#

C# code. Paste INSIDE the Detail Format event.

```
//Type cast the custom control as a TreeView
TreeView TreeView1 = new TreeView();
TreeView1 = (TreeView)this.customControl1.Control;
//Create a tree node
TreeNode objCountryTreeNode;
//Assign the text from a hidden textbox to the node
objCountryTreeNode = AddNodeToTreeView(TreeView1.Nodes, this.txtShipCountry1.Text);
//Add a second-level node
AddNodeToTreeView(objCountryTreeNode.Nodes, this.txtShipCity1.Text);
//Expand the top-level node so the second-level node is in view
objCountryTreeNode.Expand();
//Create a second top-level node
TreeNode objCustomerTreeNode;
objCustomerTreeNode = AddNodeToTreeView(TreeView1.Nodes, this.txtCustomerID1.Text);
AddNodeToTreeView(objCustomerTreeNode.Nodes, this.txtEmployeeID1.Text);
objCustomerTreeNode.Expand();
```

To load the report in the Viewer, change the Form Load event

To write code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the Form Load event

```
Dim ar = New SectionReport1()
ar.Run(False)
viewer1.LoadDocument(ar.Document)
```

To write code in C#

C# code. Paste INSIDE the Form Load event

```
var ar = new SectionReport1();
ar.Run(false);
viewer1.LoadDocument(ar.Document);
```

Section Report Scenarios

See step-by-step instructions for creating commonly used reports with Section Layout.

In this section

Create Top N Reports
: Learn to display top 10 data on a report.

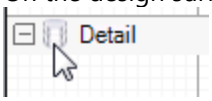
Create a Summary Report
: Learn to display summary data on a report.

Create Green Bar Reports
Learn to alternate background colors on the detail section.

Create Top N Reports

In a section report, in order to display only the top N number of details on a report, you can manipulate the data pulled by your SQL query.

To set an access data source to pull Top N data

1. On the design surface, click the **DataSource Icon** in the detail section band to open the Report Data Source dialog.


2. On the OLE DB tab of the Report Data Source dialog, next to Connection String, click the **Build** button.
3. In the Data Link Properties window that appears, select Microsoft Jet 4.0 OLE DB Provider and click the **Next** button.
4. Click the ellipsis (...) button to browse to the NWind database. Click **Open** once you have selected the appropriate access path.

   Note: The NWIND.mdb file can be downloaded from GitHub: ..\Samples14\Data\NWIND.mdb.
5. Click **OK** to close the window and fill in the Connection String field.
6. Back in Report Data Source dialog, paste the following SQL query in the Query field to fetch **Top 10** records from the database.

SQL Query

```
SELECT TOP 10 Customers.CompanyName, Sum([UnitPrice]*[Quantity])
AS Sales
FROM (Customers INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID)
INNER JOIN [Order Details] ON Orders.OrderID = [Order Details].OrderID
GROUP BY Customers.CompanyName
ORDER BY Sum([UnitPrice]*[Quantity])
DESC
```

7. Click **OK** to return to the report design surface.

To add controls to display the Top N data

1. In the Report Explorer, expand the **Fields** node, then the **Bound** node.
2. Drag and drop the following fields onto the detail section and set the properties of each textbox as indicated.

Field	Text	Location	Miscellaneous
CompanyName	Company Name	0.5, 0	
Sales	Sales	5, 0	OutputFormat = Currency

3. Go to Preview tab, to view the result.

A report with the Top 10 companies' data similar to the following will appear in the preview.

Create a Summary Report

In a section layout, you can display totals and subtotals by modifying the summary fields of a TextBox control. Use the following steps to learn how to add totals and subtotals in a report.

These steps assume that you have already added a Section Report template in a Visual Studio project and connected it to a data source. See Quick Start and Bind Reports to a Data Source for further information.

> **Note**: These steps use the Products table from the NWind database. The NWIND.mdb file can be downloaded from GitHub: ..\Samples14\Data\NWIND.mdb.

To calculate and display subtotals in a report

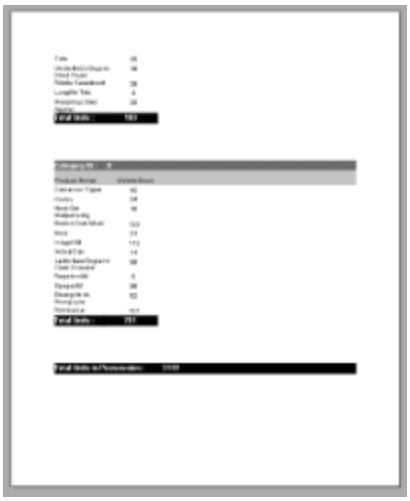
1. Right-click the design surface and select **Insert**, then **Group Header/Footer** to add group header and group footer sections to the layout.
2. With the GroupHeader section selected in the Properties Window, set its **DataField** property to *CategoryID*. This groups the data on the report according to the set field.
3. From the Report Explorer, drag and drop the following fields onto the corresponding sections of the report.
    - **Field Name** = CategoryID
      Section = GroupHeader
    - **Field Name** = ProductName
      Section = Detail
    - **Field Name** = UnitsInStock
      Section = Detail
    - **Field Name** = UnitsInStock
      Section = GroupFooter
4. With the UnitsInStock field in the GroupFooter selected, go to the Properties Window and set the following:
    - SummaryFunc: Sum
    - SummaryType: Sub Total
    - SummaryRunning: Group

    - SummaryGroup: GroupHeader1
5. Go to the Preview tab to view the report and see the **Sub Total** appear below each group of data similar to the following image.

To calculate and display grand total in a report

These steps are a continuation of the procedure above. The report generated at the end of this procedure contains totals and subtotals.

1. Right-click the design surface and select **Insert**, then **Report Header/Footer** to add report header and footer sections to the layout.
2. From Report Explorer, drag and drop the UnitsInStock field onto the ReportFooter section go to the Properties Window and set the following:
    - SummaryFunc: Sum
    - SummaryType: GrandTotal
    - SummaryRunning: All
3. Go to the Preview tab to view the report and see the **Grand Total** appear on the last page of the report.



Create Green Bar Reports

In a section report, green bar printouts can be created by setting alternate shades or background color in the report's detail section in the Format event. The following steps demonstrate how to create a Green Bar report.

1. On the design surface, double-click the detail section of the report to create an event handling method for the Detail Format event.
2. Add the following code to the handler to alternate background colors.

   To write the code in Visual Basic.NET

   Visual Basic.NET code. Paste JUST ABOVE the Detail Format event.

   ```
   Dim color As Boolean
   ```

   Visual Basic.NET code. Paste INSIDE the Detail Format event.

   ```
   If color = True Then
       Me.Detail1.BackColor = System.Drawing.Color.DarkSeaGreen
       color = False
   Else
       Me.Detail1.BackColor = System.Drawing.Color.Transparent
       color = True
   End If
   ```

   To write the code in C#

   C# code. Paste JUST ABOVE the Detail Format event.

   ```
   bool color;
   ```

   C# code. Paste INSIDE the Detail Format event.

   ```
   if(color)
   {
       this.detail.BackColor = System.Drawing.Color.DarkSeaGreen;
       color = false;
   }
   else
   {
       this.detail.BackColor = System.Drawing.Color.Transparent;
       color = true;
   }
   ```

3. Add controls like TextBox to the report design surface and preview the report.

The following image shows a Green Bar report alternating between Transparent and Dark Sea Green backgrounds:

Interactivity

Learn to perform interactive tasks in Section reports with quick how-to topics.

In this section

Add Parameters in a Section Report
: Learn how to add parameters in Section reports

Add Bookmarks
: Learn how to add bookmarks

Add Hyperlinks
: Learn how to add hyperlinks

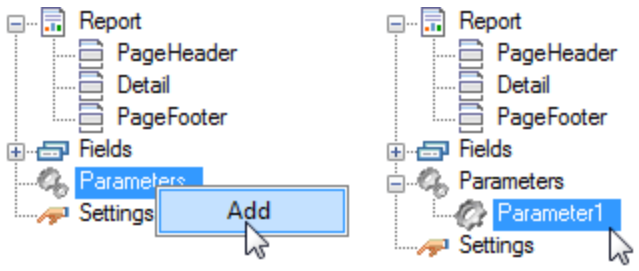
Add and Save Annotations
: Learn how to add annotations at run time and save reports containing annotations

Add Parameters in a Section Report

There are several ways to add parameters in a section report. The following sections provide a step by step overview of adding parameters in a report.

To add parameters using the Report Explorer

1. In the Report Explorer, right-click the Parameters node and select **Add**. This adds a parameter (Parameter1) as a child to the Parameters node.



2. Select the added parameter to open the Properties Window and set values in the following properties:
   - **Name**: This is the unique name of the parameter which appears as Parameter1 by default. It corresponds to the Key property in parameters entered via code.
   - **Default Value**: Sets/returns the value displayed when the user is prompted to enter a value at run time.
   - **Prompt**: Sets/returns a string displayed when a user is prompted for the value at run time.
   - **PromptUser**: Boolean value that indicates whether to prompt the user for a value or not. This is set True to use parameters at run time.
   - **Type**: This value which defaults to String defines the type of data the parameter represents. You can also set data type to Date or Boolean.
3. Pass the parameter to a field on the report, or access it programmatically as described in the run time procedure below.

To add parameters directly using a SQL query

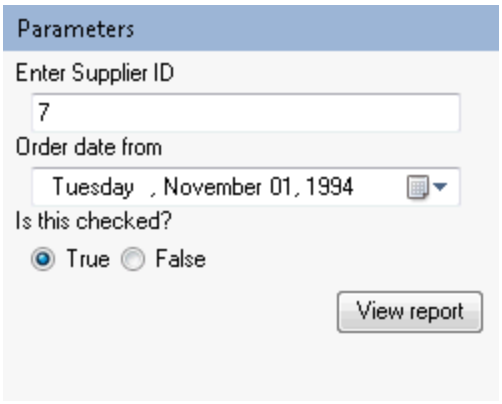
When you add SQL parameters to a report, ActiveReports displays an Enter Report Parameters dialog where the user can enter the values to fetch from the database.

1. In the detail section band, click the DataSource icon to view the Report Data Source dialog.
2. Connect the report to a data source, for example, OleDb data source. See Bind Reports to a Data Source for further details.
3. In the Query field, enter a SQL query like the one below, which contains the parameter syntax to prompt for parameter values at run time.

```
SELECT * FROM Products
INNER JOIN (Orders INNER JOIN [Order Details] ON Orders.OrderID= [Order
Details].OrderID)
ON Products.ProductID = [Order Details].ProductID
WHERE Products.SupplierID = <%SupplierID|Enter Supplier ID|7%>
AND OrderDate >= #<%OrderDate|Order date from|11/1/1994|D%>#
AND Discontinued = <%Discontinued|Is this checked?|true|B%>
```

4. Click **OK** to save the data source and return to the report design surface.

   The SQL query above causes ActiveReports to display the following dialog to the user. The user can accept these or input other values to select report data.



To add parameters at run time

You can add, edit, and delete parameters at run time. The following code demonstrates how to add a parameter and display its value in a Textbox control.

1. Double-click in the gray area below the report to create an event-handling method for the **ReportStart** event.
2. Add code to the handler to set parameters at run time.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste at beginning of code view.

```
Imports GrapeCity.ActiveReports.SectionReportModel
```

Visual Basic.NET code. Paste INSIDE the ReportStart event.

```
Dim myParam1 As New Parameter()
myParam1.Key = "myParam1"
myParam1.Type = Parameter.DataType.String

'Set to False if you do not want input from user.
myParam1.PromptUser = True
myParam1.Prompt = "Enter Data:"
myParam1.DefaultValue = "Default Value"
Me.Parameters.Add(myParam1);

'Set to True to display parameter dialog box when report is run.
Me.ShowParameterUI = True
```

To write the code in C#

C# code. Paste at beginning of code view.

```
using GrapeCity.ActiveReports.SectionReportModel;
```

C# code. Paste INSIDE the ReportStart event.

```
Parameter myParam1 = new Parameter();
myParam1.Key = "myParam1";
myParam1.Type = Parameter.DataType.String;
```

```
//Set to false if you do not want input from user.
myParam1.PromptUser = true;
myParam1.Prompt = "Enter Data:";
myParam1.DefaultValue = "Default Value";
this.Parameters.Add(myParam1);

//Set to true to display parameter dialog box when report is run.
this.ShowParameterUI = true;
```

3. In the design view, click the gray area below the report to select it and open the Properties Window.
4. Click the events icon in the Properties Window to display available events for the report.
5. Double-click FetchData. This creates an event-handling method for the report's FetchData event.
6. Add code to the handler to pass the parameter at run time.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the FetchData event.

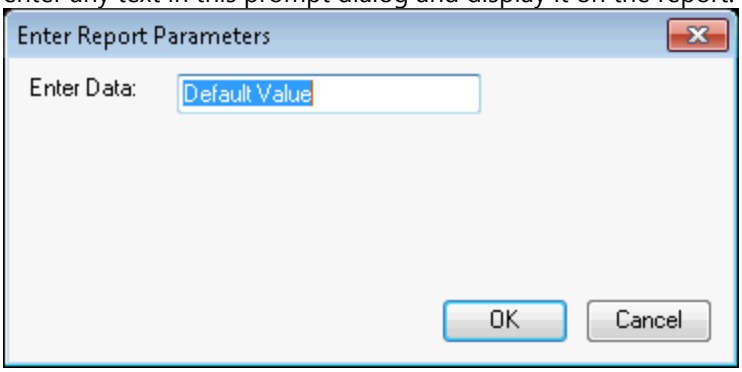
```
'Set textbox text equal to the value of the parameter.
Me.txtParam1.Text = Me.Parameters("myParam1").Value
```

To write the code in C#

C# code. Paste INSIDE the FetchData event.

```
//Set textbox text equal to the value of the parameter.
this.txtParam1.Text = this.Parameters["myParam1"].Value;
```

The run-time implementation above causes ActiveReports to display the following dialog to the user. The user can enter any text in this prompt dialog and display it on the report.



To View a Parameterized Report

The parameter prompt dialog for a parameterized report depending on how you view the report.

To get a Parameter Dialog box

1. In the Visual Studio project, add a Viewer control to the Form.
2. Double-click the Form title bar to create a Form_Load event.
3. Add the following code to the handler to view the report in the Viewer.

To write the code in Visual Basic.NET

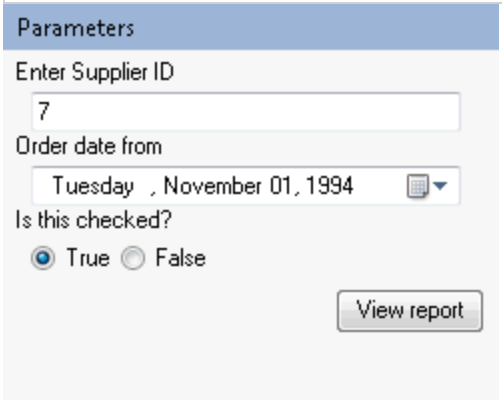
Visual Basic.NET code. Paste INSIDE the Form_Load event.

```
Dim rpt As New SectionReport1
Viewer1.Document = rpt.Document
rpt.Run()
```

To write the code in C#

C# code. Paste INSIDE the Form_Load event.

```
SectionReport1 rpt = new SectionReport1();
viewer1.Document = rpt.Document;
rpt.Run();
```



To get a Parameter Panel in the Viewer sidebar

1. In the Visual Studio project, add a Viewer control to the Form.
2. Double-click the Form title bar to create a Form_Load event.
3. Add the following code to the handler to view the report in the Viewer.

To write the code in Visual Basic.NET

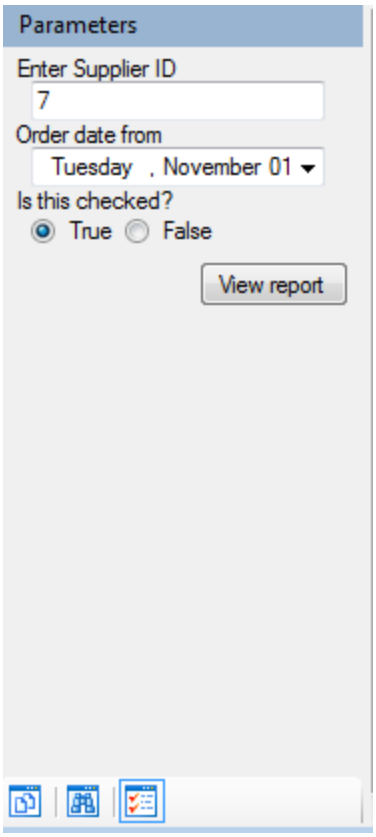
Visual Basic.NET code. Paste INSIDE the Form_Load event.

```
Dim rpt As New SectionReport1
Me.Viewer1.LoadDocument(rpt)
```

To write the code in C#

C# code. Paste INSIDE the Form_Load event.

```
SectionReport1 rpt = new SectionReport1();
viewer1.LoadDocument(rpt);
```



Add Bookmarks

In a section report, you can display bookmarks and nested bookmarks in the viewer's table of contents for fields, groups, and subreports. You can also add special bookmarks at run time.

To set up basic bookmarks

1. From the Visual Studio toolbox, drag and drop a TextBox control onto the detail section.
2. Double-click the **Detail** section of the report. This creates an event-handling method for the report's Detail_Format event.
3. Add code to the handler to set up bookmarks.

   The following example shows what the code for the method looks like.

   To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the Detail Format event.
`Me.Detail1.AddBookmark(textBox1.text)`

   To write the code in C#

C# code. Paste INSIDE the Detail Format event.
`Detail.AddBookmark(TextBox1.Text);`

To set up leveled or nested bookmarks

1. From the Report Explorer, drag and drop CustomerID and ContactName onto the detail section.
2. Double-click the **Detail** section of the report. This creates an event-handling method for the report's Detail_Format event.
3. Add code to the handler to set up bookmarks.

   The following example shows what the code to set up leveled or nested Bookmarks looks like.

   To write the code in Visual Basic.NET

   Visual Basic.NET code. Paste INSIDE the Detail Format event.

   ```
   Me.Detail1.AddBookmark(txtCustomerID.Text + "\" + txtContactName.Text)
   ```

   To write the code in C#

   C# code. Paste INSIDE the Detail Format event.

   ```
   detail.AddBookmark(txtCustomerID.Text + "\\" + txtContactName.Text);
   ```

To nest grandchild bookmarks and use bookmarks in grouping

1. From the Report Explorer, drag and drop CustomerID, ContactName and City fields onto the detail section.
2. Double-click in the **Detail** section of the report. This creates an event-handling method for the report's Detail_Format event.
3. Add code to the handler to set up a bookmark for each ContactName and nest ContactName bookmarks within each CustomerID, and CustomerID bookmarks in each City.

   The following example shows what the code for the detail section looks like.

   To write the code in Visual Basic.NET

   Visual Basic.NET code. Paste INSIDE the Detail_Format event.

   ```
   Me.Detail1.AddBookmark(txtCity.Text + "\" + txtCustomerID.Text + "\" + txtContactName.Text)
   ```

   To write the code in C#

   C# code. Paste INSIDE the Detail_Format event.

   ```
   this.detail.AddBookmark(txtCity.Text + "\\" + txtCustomerID.Text + "\\" +
   txtContactName.Text);
   ```

4. Add a GroupHeader section to the layout and set its DataField property to **City**.
5. Double-click in the Group Header section of the report. This creates an event-handling method for the report's Group Header Format event.
6. Add code to the handler to set up a bookmark for each instance of the City group.

   The following example shows what the code for the group header looks like.

   To write the code in Visual Basic.NET

   Visual Basic.NET code. Paste INSIDE the Group Header Format event.

   ```
   Me.GroupHeader1.AddBookmark(txtCity.Text)
   ```

   To write the code in C#

   C# code. Paste INSIDE the Group Header Format event.

   ```
   this.groupHeader1.AddBookmark(txtCity.Text);
   ```

To combine parent report and subreport bookmarks

1. From the Report Explorer, drag and drop the CustomerID field onto the detail section of the **Parent** report.
2. Double-click the Detail section to create an event-handling method for the report's Detail Format event.
3. Add code to the handler to create a bookmark for each instance of the CustomerID field in the main report.

   The following example shows what the code for the method looks like for the main report.

   To write the code in Visual Basic.NET

   Visual Basic.NET code. Paste INSIDE the Detail Format event of the main report.

   ```
   Me.Detail1.AddBookmark(txtCustomerID.Text)
   ```

   To write the code in C#

   C# code. Paste INSIDE the Detail Format event of the main report.

   ```
   detail1.AddBookmark(txtCustomerID.Text);
   ```

4. From the Report Explorer, drag and drop the ContactName field onto the detail section of the **Subreport**.
5. Double-click in the Detail section to create an event-handling method for the report's Detail Format event.
6. Add code to the handler to create a bookmark for each instance of the ContactName field in the subreport.

   The following example shows what the code for the method looks like for the subreport.

   To write the code in Visual Basic.NET

   Visual Basic.NET code. Paste INSIDE the Detail Format event of the subreport.

   ```
   Me.Detail1.AddBookmark(CType(Me.ParentReport.Sections("Detail1").Controls("txtCustomerID"),
   TextBox).Text
   + "\" + Me.txtContactName.Text)
   ```

   To write the code in C#

   C# code. Paste INSIDE the Detail Format event of the subreport.

   ```
   this.detail1.AddBookmark(((TextBox)
   (this.ParentReport.Sections["Detail1"].Controls["txtCustomerID"])).Text
   + "\\" + this.txtContactName.Text);
   ```

To add special bookmarks at run time

To create and add special bookmarks to the bookmarks collection at run time, add the bookmarks to the report document's pages collection.

> ⚠ **Caution**: Remember that the page collection does not exist until the report runs, so use this code in the ReportEnd event or in form code after the report has run.

To write the code in Visual Basic.NET

1. Click in the gray area outside the report and right-click to select Properties from the context menu.
2. Click the Events icon in the Properties Window to display events available for the report.
3. Double-click **ReportEnd**. This creates an event-handling method for the ReportEnd event.
4. Add code to the handler to add a bookmark.

The following example shows what the code for the method looks like.

Visual Basic.NET code. Paste INSIDE the ReportEnd event.

```
Me.Document.Pages(0).AddBookmark("New Bookmark", 1)
```

To write the code in C#

C# code. Paste INSIDE the ReportEnd event.

```
this.Document.Pages[0].AddBookmark("New Bookmark", 1);
```

To view a report bookmarks in the Viewer or Preview tab

1. Add the ActiveReports Viewer control to your Windows Form.
2. Add code to display the report in the Viewer. See Preview Reports for further information.
3. Press **F5** to run the report.
4. On the Viewer toolbar, select Toggle Sidebar to open the sidebar and click the **Document Map** button to view the list of bookmarks.

Add Hyperlinks

In a section report, you can add hyperlinks in a report using the **Hyperlink** property available with the following controls:

- Label
- TextBox
- Picture

You can add hyperlinks that connect to a Web page, open an e-mail, or jump to a bookmark.

Note: Specify the full URL address (for example, "http://www.grapecity.com") for the **Hyperlink** property to avoid broken links while viewing reports.

To link to a Web page

1. Select an existing control or drag and drop a control from the Visual Studio toolbox onto the design surface.
2. Right-click the control to open the Properties Window.
3. In the Properties Window, set the **HyperLink** property to any valid URL. For example, for example, http://www.grapecity.com.

To link to an e-mail address

1. Select an existing control or drag and drop a control from the Visual Studio toolbox onto the design surface.
2. Right-click the control to open the Properties Window.
3. In the Properties Window, set the **HyperLink** property to mailto: *any valid e-mail address*.

To parse the URL out of a database field for a hyperlink

1. From the Report Explorer, drag and drop the link field onto the design surface.
2. Double-click the section where you had placed the link field. This creates an event-handling method for the section's **Format** event.
3. Add code to the Format event to,

- Parse the URL out of the **Link** field
- Assign it to the **HyperLink** property of **TextBox**
- Remove the URL markers from the text displayed in **TextBox**

The following example shows what the code for the method looks like.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the Format event.

```
Dim iStart As Integer
Dim sHTML As String
If textBox1.Text <> "" Then
    iStart = InStr(1, textBox1.Text, "#", CompareMethod.Text)
    sHTML = Right(textBox1.Text, (Len(textBox1.Text) - iStart))
    sHTML = Replace(sHTML, "#", "", 1, -1, CompareMethod.Text)
    textBox1.HyperLink = sHTML
    textBox1.Text = Replace(textBox1.Text, "#", "", 1, -1, CompareMethod.Text)
End If
```

To write the code in C#

C# code. Paste INSIDE the Format event.

```
int iStart;
string sHTML;
if (textBox1.Text != "")
    {
     iStart = textBox1.Text.IndexOf("#",0);
     sHTML = textBox1.Text.Substring(iStart, textBox1.Text.Length - iStart);
     sHTML = sHTML.Replace("#", "");
     textBox1.HyperLink = sHTML;
     textBox1.Text = textBox1.Text.Replace("#", "");
    }
```

To create a hyperlink that jumps to a bookmark

1. From the Report Explorer, drag and drop a field onto the design surface.
2. Double-click the section where you had placed the field. This creates an event-handling method for the section's **Format** event.
3. Add the following code inside the Format event.

The following example shows what the code for the method looks like.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste JUST ABOVE the Format event.

```
Public pBM As New BookmarksCollection()
Dim iEntry As Integer
```

Visual Basic.NET code. Paste INSIDE the Format event.

```
Me.Detail1.AddBookmark(Me.textBox1.Text)
Me.txtEntry.HyperLink = "toc://" + pBM(iEntry - 1).Label
Me.txtEntry.Text = pBM(iEntry - 1).Label
Me.txtPage.Text = pBM(iEntry - 1).PageNumber
```

To write the code in C#

C# code. Paste JUST ABOVE the Format event.

```
public BookmarksCollection pBM = new BookmarksCollection();
int iEntry;
```

C# code. Paste INSIDE the Format event.

```
this.detail.AddBookmark(this.textBox.Text);
this.txtEntry.HyperLink = "toc://" + pBM[iEntry - 1].Label;
this.txtEntry.Text = pBM[iEntry - 1].Label;
this.txtPage.Text = pBM[iEntry - 1].PageNumber.ToString();
```

To display the page number of the bookmark in the table of contents

1. Select the gray area outside the report and right-click to choose Properties option from the context menu.
2. In the Properties Window that appears, click the Events button to get the list of events for the report.
3. Select the **FetchData** event from that list and double-click it. This creates an event-handling method for the report's FetchData event in the code behind.
4. Add code to the handler to retrieve information to populate the report fields.

The following example shows what the code for the method looks like.

To write the code in Visual Basic

Visual Basic.NET code. Paste INSIDE the FetchData event.

```
If iEntry > pBM.Count - 1 Then
    eArgs.EOF = True
Else
    eArgs.EOF = False
    iEntry += 1
End If
```

To write the code in C#

C# code. Paste INSIDE the FetchData event.

```
if (iEntry > pBM.Count - 1)
{
    eArgs.EOF = true;
}
else
{
    eArgs.EOF = false;
```

```
    iEntry += 1;
}
```

Add and Save Annotations

In a section report, you can save a report containing annotations along with the report data into an RDF file. You can also add annotations at run time. The following steps demonstrate how to accomplish these tasks in code.

These steps assume that you have already added a Section Report (code based) template in a Visual Studio project. See Quick Start for more information.

To save annotations

The following example shows how to add a **Save Annotated Report** button to the viewer and save a report with annotations in RDF.

1. From the Visual Studio toolbox, drag a **Button** control onto the viewer.
2. Set the **Text** property of the button to **Save Annotated Report**.
3. Double-click the button. This creates an event-handling method for the button **Click** event.
4. Add code to the click handler to save the document to an **RDF** file. See Save and Load RDF Report Files for more information on loading the saved RDF file into the viewer.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the button Click event.

```
Me.Viewer1.Document.Save("C:\UserAnnotations.rdf")
```

To write the code in C#

C# code. Paste INSIDE the button Click event.

```
this.viewer1.Document.Save("C:\\UserAnnotations.rdf");
```

To add annotations in code

The following example shows how to add annotations at run time and save the report data and annotations to an RDF file.

1. Double-click the title bar of the Form in which you host the viewer. This creates an event-handling method for the Form_Load event.
2. Add code to the handler to run the report, add annotations, display the report in the viewer, and save it into an RDF file.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste ABOVE the class.

```
Imports GrapeCity.ActiveReports.Document.Section.Annotations
```

Visual Basic.NET code. Paste INSIDE the Form Load event.

```
   Dim rpt As New SectionReport1
```

```
'Run the report first.
rpt.Run()

'Assign the viewer.
Me.Viewer1.Document = rpt.Document

'Create an annotation and assign property values.
Dim circle As New AnnotationCircle
circle.Color = System.Drawing.Color.GreenYellow
circle.Border.Color = System.Drawing.Color.Chartreuse

'Add the annotation.
circle.Attach(1,1) 'screen location
Me.Viewer1.Document.Pages(0).Annotations.Add(circle)

'Set the size properties. The annotation must be added to the page first.
circle.Height = 0.25
circle.Width = 0.50

'Save annotations with the report in an RDF file.
rpt.Document.Save("C:\AnnotatedReport.rdf")
```

To write the code in C#

C# code. Paste ABOVE the class.

```
using GrapeCity.ActiveReports.Document.Section.Annotations;
```

C# code. Paste INSIDE the Form Load event.

```
SectionReport1 rpt = new SectionReport1();
//Run the report first.
rpt.Run();

//Assign the viewer
this.viewer1.Document = rpt.Document;

//Create an annotation and assign property values.
AnnotationCircle circle = new AnnotationCircle();
circle.Color = System.Drawing.Color.GreenYellow;
circle.Border.Color = System.Drawing.Color.Chartreuse;

//Add the annotation.
circle.Attach(1,1); //screen location
this.viewer1.Document.Pages[0].Annotations.Add(circle);

//Set the size properties. The annotation must be added to the page first.
circle.Height = 0.25f;
circle.Width = 0.50f;
```

```
//Save annotations with the report in an RDF file.
rpt.Document.Save("C:\\AnnotatedReport.rdf");
```

Common Tasks

Learn to perform common tasks in Section reports with quick how-to topics.

In this section

Inherit a Report Template
: Learn how to inherit other reports from a base report as a template

Change Ruler Measurements
: Learn how to change ruler measurements

Conditionally Show or Hide Details
: Learn how to use conditions to control the display of report's detail section

Use External Style Sheets
: Learn how to set custom style values in the Styles page

Insert or Add Pages
: Learn how to add and insert pages from one report to another

Add Groups
: Learn how to add group data by adding a pair of group header and group footer sections to the report

Embed Subreports
: Learn how to embed a subreport into a parent report

Add Code to Layouts Using Script
: Learn how to use script to access controls, functions in a class, namespaces, etc.

Save and Load RDF Report Files
: Learn how to save and load RDF report files

Save and Load RPX Report Files
: Learn how to save and load RPX report files

Print Multiple Copies, Duplex and Landscape
: Learn how to modify various printer settings

Inherit a Report Template

In a section layout, you can create a base report as a template from which other reports can inherit. This behavior is similar to creating a master report and is available in a Section Report (code-based) layout.

Inheriting reports is useful when multiple reports share common features, such as identical page headers and footers. Instead of recreating the look every time, create template headers and footers once and use inheritance to apply them to other reports.

Use the following instructions to create a base report and inherit it in other reports.

> ⚠ **Caution:** Base reports and the reports that inherit from them cannot contain controls with duplicate names. You can compile and run your project with duplicate control names, but you cannot save the layout until you change the duplicate names.

To create a base report

1. In a Visual Studio project, add a Section Report (code-based) and name it **rptLetterhead**. See Quick Start for more

information.

2. In the report template that appears, add the following controls from the Visual Studio toolbox to the indicated section of rptLetterhead and set the properties.

Control	Section	Location	Size	Miscellaneous
Picture	PageHeader	0, 0 in	3, 0.65 in	Image = (click ellipsis and navigate to the location of your ima PictureAlignment = TopLeft
Label	PageHeader	1.16, 0.65 in	1.8, 0.25 in	Text = Inheritance Font = Arial, 15pt, style=Bold
Label	PageFooter	0, 0 in	6.5, 0.19 in	Text = https://www.grapecity.com HyperLink = https://www.grapecity.com Font/Bold = True Alignment = Center

3. Right-click the gray area below the design surface and choose properties, to open the Properties window.
4. In the Properties window, set the **MasterReport** property to **True**. Setting the MasterReport property to True locks the Detail section.

> ⚠ **Caution:** Do not set the **MasterReport** property to **True** until you have finished designing or making changes to the report. Setting this property to True triggers major changes in the designer file of the report.

You can use the Page Header and Page Footer sections to design the base report. When you create reports that inherit the layout from this base report, only the detail section is available for editing.

To inherit layout from a base report

These steps assume that you have already added another Section Report (code-based) template. This report functions like a content report where you can create the layout of the Detail section.

1. In a Visual Studio project, add a Section Report (code-based) and name it **rptLetter**.
2. In the Solution Explorer, right-click the new report and select the **View Code** option to open the code behind of the report.
3. In the code view, modify the inheritance statement as shown below. The content report inherits from the base report instead of **GrapeCity.ActiveReports.SectionReport**.

> ⚠ **Caution**: The existing report layout in the content report is lost once you inherit the base report. Even if you change it back to GrapeCity.ActiveReports.SectionReport, the original layout in content report will not be retrieved.

To write the code in Visual Basic.NET

Visual Basic.NET code. Replace *YourContentReport*, *YourProjectName* and *YourMasterReportName* with relevant names.

```
Partial Public Class rptLetter Inherits YourProjectName.rptLetterhead
```

To write the code in C#

C# code. Replace *YourContentReport*, *YourProjectName* and *YourMasterReportName* with relevant names.

```
public partial class rptLetter : YourProjectName.rptLetterhead
```

4. Close the reports and from the Build menu on the Visual Studio menu bar, select **Rebuild**. When you reopen the report, the inherited sections and controls are disabled.

Note: To apply further changes from the base report to the content report, you might have to rebuild the project again.

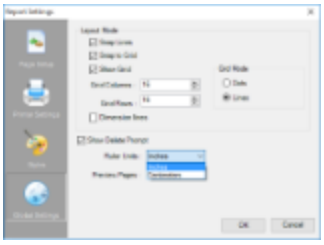
Change Ruler Measurements

In a section layout, you can change ruler measurements from inches to centimeters and centimeters to inches. Use the following instructions to modify ruler measurements at design- time and run-time.

To change ruler measurements at design-time

At design time, you can change the ruler measurements from the Report Settings Dialog.

1. In the Report Explorer, double-click the **Settings** node.
2. In the Report Settings dialog that appears, click **Global Settings**.
3. From the **Ruler Units** dropdown select Centimeters or Inches.



To call a measurement conversion at run-time

Call the **CmToInch ('CmToInch Method' in the on-line documentation)** method or **InchToCm ('InchToCm Method' in the on-line documentation)** method at run-time to change measurements. For example, you can use the following code when you are working in centimeters and need to convert a Label's position measurements from centimeters to inches at run-time.

1. On the design surface select the section containing a control like a Label.
2. In the Properties Window, click the Events button to get a list of report events.
3. Select the **Format** event and double-click to create an event-handling method.
4. Add code like the following to the handler to set the size of the control using centimeters to inches.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste inside the Format event.

```
Me.Label1.Left = SectionReport1.CmToInch(2)
Me.Label1.Top = SectionReport1.CmToInch(2)
```

To write the code in C#

C# code. Paste inside the Format event.

```
this.label1.Left = SectionReport1.CmToInch(2);
this.label1.Top = SectionReport1.CmToInch(2);
```

Conditionally Show or Hide Details

In a section layout, you can use conditions in the Format event to control the display of report's detail section at run time.

These steps assume that you have already added a Section Report (code-based) template in a Visual Studio project and connected it to a data source. See Quick Start and Bind Reports to a Data Source for further information.

Note: These steps use the Products table from the NWind database. The NWIND.mdb file can be downloaded from GitHub: ..\Samples14\Data\NWIND.mdb.

1. From the Report Explorer, drag and drop the following fields onto the detail section of the report and set their properties in the Properties Window.

Field Name	Properties
ProductName	**Location**: 0, 0.104 in **Size**: 2.667, 0.2 in
Discontinued	**Location**: 2.667, 0.104 in **Size**: 2.021, 0.2 in
ReorderLevel	**Location**: 4.688, 0.104 in **Size**: 1.812, 0.2 in

2. Double-click the detail section of the report to create an event-handling method for the Format event.
3. Add the following code to the handler to hide the details of a product which is discontinued.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the Detail_Format event.

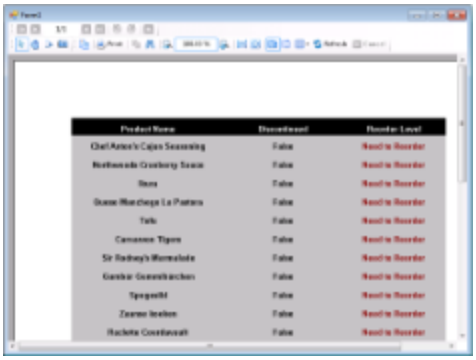
```
If Me.txtReorderLevel1.Value = 0 And Me.txtDiscontinued1.Value = False Then
    Me.Detail1.Visible = True
    Me.txtReorderLevel1.Text = "Need to Reorder"
    Me.txtReorderLevel1.ForeColor = System.Drawing.Color.DarkRed
Else
    Me.Detail1.Visible = False
End If
```

To write the code in C#

C# code. Paste INSIDE the detail_Format event.

```
if (int.Parse(txtReorderLevel1.Value.ToString()) == 0 && txtDiscontinued1.Text ==
"False")
{
this.detail1.Visible = true;
this.txtReorderLevel1.Text = "Need to Reorder";
this.txtReorderLevel1.ForeColor = System.Drawing.Color.DarkRed;
}
else
{
this.detail1.Visible = false;
}
```

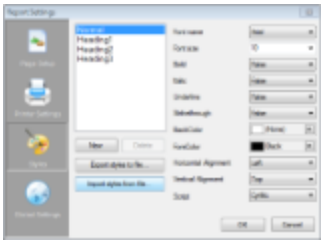
4. In Form1 of the Visual Studio project, add a Viewer control and load the report you created above in it. See Windows Forms Viewer for further details.
5. Press F5 to debug and see a report with discontinued products hidden from view.



Use External Style Sheets

In a section layout, you can set custom style values in the Styles page of the Report Settings Dialog, and then apply the styles to controls using the ClassName property from the Properties Window.

You can also apply these same styles to controls in other reports, by exporting them to a XML file of type *.reportstyle and selecting it in other reports using the Report Settings dialog.



Note: You can apply styles to the CheckBox, Label, TextBox, and ReportInfo controls only.

To modify or create a style and save it to an external style sheet

1. In the Report Explorer, right-click the Settings node and select **Show**.
2. In the **Report Settings** dialog that appears, go to the Styles page. On the Styles page, there are four predefined styles: Normal, Heading1, Heading2 and Heading3.
3. Click any of these styles in the list to modify them using the fields on the right, or click the **New** button to add a new style.
4. Click the **Export styles to file** button to save the existing styles in a style sheet.
5. In the **Save As** dialog that appears, navigate to the location where you want to save the style sheet, provide an name for the file and click the **Save** button to save it as an external *.reportstyle file.
6. In the Report Settings dialog, click the **OK** button to close the dialog and save the styles in the current report.

To load and apply an external style sheet at design time

1. In the Report Explorer, right-click the Settings node and select **Show**.
2. In the **Report Settings** dialog that appears, go to the Styles page.
3. On the Style page, click the **Import styles from file** button.
4. A message box warns that current styles will be deleted. Click **Yes** to continue.
5. In the Open dialog that appears, navigate to the *.reportstyle file that you want to use and click the **Open** button to load the external style sheet.
6. On the design surface, select the control you want to apply the style to and right-click to choose Properties.
7. In the Properties Window, from the **Class Name** property drop down select a style to apply (like Heading1).

To load an external style sheet at run time and apply it

1. Right-click the gray area outside the design surface and select Properties.
2. In the Properties Window that appears, click the Events button. A list of report events appear.
3. Select the ReportStart event and double click to create an event-handling method.
4. Add the following code to the handler to load an external style sheet.

Visual Basic.NET code. Paste INSIDE the ReportStart event.

```
Me.LoadStyles("C:\MyStyleSheet.reportstyle")
```

C# code. Paste INSIDE the ReportStart event.

```
this.LoadStyles(@"C:\MyStyleSheet.reportstyle");
```

To apply a style to a control at run time

The following steps assume that you have already loaded an external style sheet to the report.

1. On the design surface select the section containing the control.
2. In the Properties Window, click the Events button. A list of report events appear.
3. Select the **Format** event and double click to create an event-handling method.
4. Add the following code to the handler to apply a style to a control.

Visual Basic.NET code. Paste INSIDE the Format event.

```
Me.TextBox.ClassName = "Heading1"
```

C# code. Paste INSIDE the Format event.

```
this.textBox.ClassName = "Heading1";
```

Insert or Add Pages

In a section layout, you can run multiple reports, merge their entire page collection or specific portions and view it as a single report. You can save the document containing merged reports to an RDF file or even export them.

These steps assume that you have already placed a Viewer control on a Windows Form and your Visual Studio project contains two section layout (code based) reports (rptOne and rptTwo). See Quick Start and Using the Viewer for more information.

To add pages from one report to another

To add an entire report to another, use code like the one in the example below to iterate through the entire pages collection of a report and append it to the first report. The **Add ('Add Method' in the on-line documentation)** of the PagesCollection takes one parameter (value), which references a report document page.

1. In the design view of the Form containing the Viewer, double-click the title bar of the Form to create an event-handling method for the **Form_Load** event.
2. Add the following code to the handler to add the entire rptTwo page collection to rptOne.

   The following example shows what the code for the Add() method looks like.

   To write the code in Visual Basic.NET

   Visual Basic.NET code. Paste INSIDE the Form Load event.

```
Dim i As Integer
Dim rpt As New rptOne()
rpt.Run()
Dim rpt2 As New rptTwo()
rpt2.Run()
For i = 0 To rpt2.Document.Pages.Count - 1
rpt.Document.Pages.Add(rpt2.Document.Pages(i))
Next
Viewer1.Document = rpt.Document
```

   To write the code in C#

   C# code. Paste INSIDE the Form Load event.

```
int i;
rptOne rpt1 = new rptOne();
rpt1.Run();
rptTwo rpt2 = new rptTwo();
rpt2.Run();
for(i = 0; i < rpt2.Document.Pages.Count; i++)
{
rpt1.Document.Pages.Add(rpt2.Document.Pages[i]);
}
```

```
viewer1.Document = rpt1.Document;
```

To add a range of pages from one report to another

To add a range of pages from one report to another, use the **AddRange ('AddRange Method' in the on-line documentation)**. This method has two overloads, each with one parameter. The first overload takes an array of page objects which you can use to append only the specified pages from the second report onto the first (as in the example below).

1. In the design view of the Form containing the Viewer, double-click the title bar of the Form to create an event-handling method for the **Form_Load** event.
2. Add the following code to the handler to use the AddRange() method to add pages from rptTwo to rptOne.

   The following example shows what the code for the AddRange() method looks like.

   To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the Form Load event.

```
Dim rpt1 As New rptOne()


rpt1.Run()


Dim rpt2 As New rptTwo()


rpt2.Run()


rpt1.Document.Pages.AddRange(New GrapeCity.ActiveReports.Document.Section.Page()
{rpt2.Document.Pages(1), rpt2.Document.Pages(2)})


Viewer1.Document = rpt1.Document
```

   To write the code in C#

C# code. Paste INSIDE the Form Load event.

```
rptOne rpt1 = new rptOne();
rpt1.Run();
rptTwo rpt2 = new rptTwo();
rpt2.Run();
rpt1.Document.Pages.AddRange(new GrapeCity.ActiveReports.Document.Section.Page[]
{rpt2.Document.Pages[0],rpt2.Document.Pages[1]} );
viewer1.Document = rpt1.Document;
```

To insert pages from one report into another

To insert pages from one report to another, use the **Insert ('Insert Method' in the on-line documentation)** that takes two parameters, an index, which determines where to insert the pages in the main report, and a value which references the report page to insert.

1. In the design view of the Form containing the Viewer, double-click the title bar of the Form to create an event-handling method for the **Form_Load** event.
2. Add the following code to the handler to insert page 1 of rptTwo at the beginning of rptOne.

   The following example shows what the code for the Insert() method looks like.

   To write the code in Visual Basic.NET

   Visual Basic.NET code. Paste INSIDE the Form Load event.

```
Dim rpt1 As New rptOne()


rpt1.Run()


Dim rpt2 As New rptTwo()


rpt2.Run()


rpt1.Document.Pages.Insert(0, rpt2.Document.Pages(0))


Viewer1.Document = rpt1.Document
```

   To write the code in C#

   C# code. Paste INSIDE the Form Load event.

```
rptOne rpt1 = new rptOne();


rpt1.Run();


rptTwo rpt2 = new rptTwo();


rpt2.Run();


rpt1.Document.Pages.Insert(0, rpt2.Document.Pages[0]);
```

```
viewer1.Document = rpt1.Document;
```

To insert a new page at a specific report location

To insert a new blank page at a specific location in the report, use the **InsertNew ('InsertNew Method' in the on-line documentation)** which takes one parameter, **index**, which specifies the page after which you want to insert a new blank page.

1. In the design view of the viewer form, double-click the title bar of the Form to create an event-handling method for the **Form_Load** event.
2. Add the following code to the handler to insert a blank page at the beginning of rptOne.

   The following example shows what the code for the InsertNew() method looks like.

   To write the code in Visual Basic.NET

   Visual Basic.NET code. Paste INSIDE the Form Load event.

```
Dim rpt1 As New rptOne()


rpt1.Run()


rpt1.Document.Pages.InsertNew(0)


Viewer1.Document = rpt1.Document
```

   To write the code in C#

   C# code. Paste INSIDE the Form Load event.

```
rptOne rpt1 = new rptOne();


rpt1.Run();


rpt1.Document.Pages.InsertNew(0);


viewer1.Document = rpt1.Document;
```

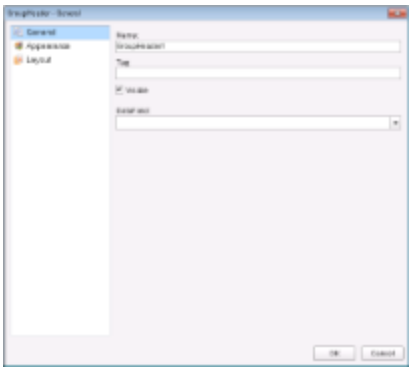
Add Groups

In a section report, you can group data by adding a pair of group header and group footer sections to the report. These sections appear immediately above and below the detail section.

> ⚠ **Caution**: You cannot add a header section without a corresponding footer section. If you try to do so in code, the results are unstable.

You can set the properties for the GroupHeader and GroupFooter sections in their corresponding dialogs. Following is a list of properties you can set through the options in these dialogs. Each option in the GroupHeader dialog corresponds to a property in the properties window. To access the properties directly, select the section and open the properties window. See the associated property names in parenthesis with each dialog option below.

GroupHeader Dialog



To access the GroupHeader dialog, right click the group header and in the Properties window under the properties list where the commands are displayed, click the **Property dialog** link. See Properties Window for further information on commands.

General

- **Name** (Name)**:** Indicates the name of the GroupHeader in code. It is unique for a report.
- **Tag** (Tag): Indicates the user-defined information persisted with the GroupHeader section.
- **Visible** (Visible): Checkbox to specify the visibility of the GroupHeader section.
- **DataField** (DataField): Field or expression on which you group the data.

Appearance

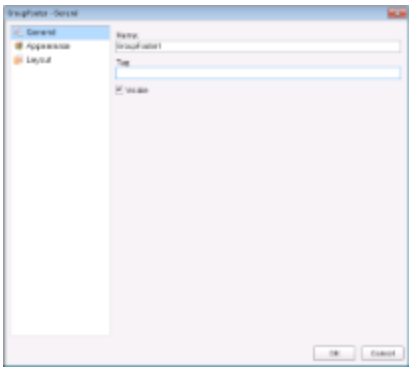
- **Background color** (BackColor): Dropdown list to set the background color of the GroupHeader section.

Layout

- **Insert new page** (NewPage): Dropdown list to determine whether a new page is inserted before and/or after displaying the GroupHeader section.
- **Insert new column** (NewColumn): Dropdown list to determine whether a new column (in a multi-column report) appears before and/or after displaying the GroupHeader section.
- **Repeat section** (RepeatStyle): Dropdown list to specify whether the GroupHeader section appears with every column or page that the Detail section or associated footer appears on.
- **Keep section and its footer on a single page** (GroupKeepTogether): Dropdown list to specify whether the GroupHeader section and its footer appear as a single block on the same page or not.
- **Keep section on a single page** (KeepTogether): Checkbox to specify whether the GroupHeader section appears on a single page.
- **Keep section and its footer in a single column** (ColumnGroupKeepTogether): Checkbox to specify whether the GroupHeader section and its footer appear as a single block in the same column.
- **Keep section underneath the following section** (UnderlayNext): Checkbox to specify whether the GroupHeader

section appears in the following section or not. It allows you to show group header information inside the group details, so long as you keep the BackColor property of the Detail section set to Transparent.
- **Use column layout** (ColumnLayout): Checkbox to determine whether the GroupHeader section uses the same column layout as the Detail section.
- **Can increase to accommodate contents** (CanGrow): Checkbox to specify whether the height of the GroupHeader section can grow when its controls extend beyond its original height.
- **Can decrease to accommodate contents** (CanShrink): Checkbox to specify whether the height of the GroupHeader section can adjust to the total height of controls placed in it.

GroupFooter Dialog



To access the GroupFooter dialog, right click the group footer and in the Properties window under the properties list where the commands are displayed, click the **Property dialog** link. See Properties Window for further information on commands.

General

- **Name** (Name): Indicates the name of the GroupFooter in code. It is unique for a report.
- **Tag** (Tag): Indicates the user-defined information persisted with the GroupFooter section.
- **Visible** (Visible): Checkbox to specify the visibility of the GroupFooter section.

Appearance

- **Background color** (BackColor): Dropdown list to set the background color of the GroupFooter section.

Layout

- **Insert new page** (NewPage): Dropdown list to determine whether a new page is inserted before and/or after displaying the GroupFooter section.
- **Insert new column** (NewColumn): Dropdown list to determine whether a new column (in a multi-column report) appears before and/or after displaying the GroupFooter section.
- **Keep section on a single page** (KeepTogether): Checkbox to determine whether the GroupFooter section appears on a single page.
- **Use column layout** (ColumnLayout): Checkbox to specify whether the GroupFooter section uses the same column layout as the Detail section.
- **Print at the bottom of page** (PrintAtBottom): Checkbox to specify whether the GroupFooter section is printed at the bottom of the page immediately before the PageFooter section.
- **Can increase to accommodate contents** (CanGrow): Checkbox to specify whether the height of the GroupFooter section can grow when its controls extend beyond its original height.
- **Can decrease to accommodate contents** (CanShrink): Checkbox to specify whether the height of the GroupFooter

section can adjust to the total height of controls placed in it.

When you run the report, it renders the group header, followed by all related instances of the detail section, and then the group footer. It renders a new group header section for each instance of the grouping field.

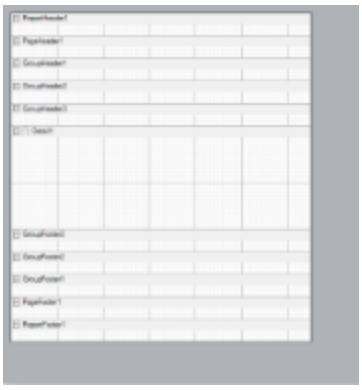
Controls in the group header render once for each instance of the group, so you can place the column header labels to describe the data in the detail fields here.

Multiple Grouping

In a section report, you can nest group header and footer pairs and group each on a different field. You can add up to 32 groupings in one report.

> **Note**: As with any group header and footer pair, group your data on the fields that you specify in the **DataField ('DataField Property' in the on-line documentation)** property of the group header, but in the order of your groups. For example: `SELECT * FROM Customers ORDER BY GroupHeader1DataField, GroupHeader2DataField, GroupHeader3DataField`

See the image below for the order in which report sections appear on the report. GroupHeader1 in the image was added first and appears above the other two group headers, while its pair GroupFooter1, appears below the other two group footers.



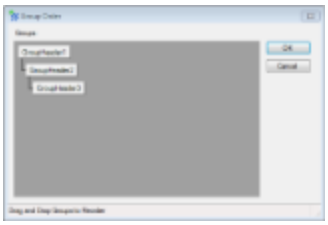
When you run a report with multiple groupings like the one above, the sections print in the following order:

1. **ReportHeader1** prints once and does not repeat.
2. **PageHeader1** prints once at the top of each page.
3. **GroupHeader1** prints once for the first value its DataField returns.
4. **GroupHeader2** prints once for the first value its DataField returns within the context of GroupHeader1's DataField value.
5. **GroupHeader3** prints once for the first value its DataField returns within the context of GroupHeader2's DataField value.
6. **Detail1** prints once for each record that falls within the context of GroupHeader3's DataField value.
7. **GroupFooter3** prints once at the end of the records that fall within the context of GroupHeader3's DataField value.
8. **GroupHeader3** may print again, if more values return within the context of GroupHeader2's DataField value.
9. Each time GroupHeader3 prints again, it is followed by Detail1 (once for each related record) and GroupFooter3.
10. **GroupFooter2** prints once after GroupFooter3.
11. **GroupHeader2** may print again, if more values return within the context of GroupHeader1's DataField value.
12. Each time GroupHeader2 prints again, it is followed by Detail1 (once for each related record) and GroupFooter2.
13. **GroupFooter1** prints once after GroupFooter2.

14. **GroupHeader1** prints once for the second value its DataField returns, followed by GroupHeader2, and so on in a pattern similar to the one above.
15. **ReportFooter1** prints once on the last page where the data displayed in the report ends.
16. **PageFooter1** prints once at the bottom of each page. Also, its position within groups varies.

    > **Note**: At design time, although the PageFooter section is located above the ReportFooter section, at run time it appears after the ReportFooter section on the last page.

With many groupings, you might find the need to rearrange the order of your groups. If your report has more than one group, you can right-click the report surface, and select **Reorder Groups**. This opens the **Group Order** dialog, where you can drag the groups and set them in any order you want.



Alternatively, you can also click the **Reorder Groups** button in the ActiveReports toolbar, to open the Group Order dialog. See Toolbar for further information.

In a section report, you can set grouping on a field or a field expression. Use the following steps to understand grouping in a section report.

These steps assume that you have already added a Section Report (xml-based) or Section Report (code based) template and connected it to a data source. See Quick Start for further information.

1. Right-click the design surface of a report and select **Insert**, then **Group Header/Footer**. Group Header and Footer sections appear immediately above and below the detail section.
2. With the GroupHeader section selected, go to the Properties window and set the **DataField ('DataField Property' in the on-line documentation)** to a field on which you want to group the data. For example, Country from Customers table in the NWind database.

   > **Note:** You can also set a field expression in the DataField property. For example, `=Country + City`.
3. Drag and drop the grouping field onto the GroupHeader section to see the grouping field while previewing the report.
4. Drag and drop data fields onto the detail section. All the data placed inside the detail section gets grouped according to grouping field.
5. Preview the report to see the result.

The following image shows a customer list grouped on the Country field.

> **Tip:** In a section report, data is grouped in the order in which it is fetched in the raw form. Therefore, for best results, while setting the SQL query in your report data source, order the data by the grouping field. For example., `SELECT * FROM Customers ORDER BY Country`

Embed Subreports

To embed a subreport into a parent report, you add two reports (one parent and one child report) to a Visual Studio project, and from the ActiveReports 14 Section Report toolbox, drag the SubReport control onto the parent report. The following steps take you through the process of adding a subreport in a section report.

These steps assume that you have already added a Section Report (code-based) template in a Visual Studio project. See Quick Start for further information.

To add code to create an instance of the child report in the parent report

1. Double-click the gray area around the parent report to create an event-handling method for the **ReportStart** event.
2. Add code like the following to the handler to create a new instance of the child report.

To write the code in Visual Basic

Visual Basic.NET code. Paste JUST ABOVE the ReportStart event.
`Dim rpt As rptYourChildReportName`
Visual Basic.NET code. Paste INSIDE the ReportStart event.
`rpt = New rptYourChildReportName()`

To write the code in C#

C# code. Paste JUST ABOVE the ReportStart event.
`private rptYourChildReportName rpt;`
C# code. Paste INSIDE the ReportStart event.
`rpt = new rptYourChildReportName();`

> ⚠ **Caution**: It is recommended that you do not create a new instance of the subreport in the **Format** event. Doing so creates a new subreport each time the section Format code is run, using a lot of memory.

To add code to display the child report in a subreport control on a parent report

1. Add the SubReport control onto the design surface of the parent report.
2. Double-click the detail section of the report to create a detail_Format event.
3. Add code like the following to the handler to display a report in the SubReport control.

To write the code in Visual Basic

Visual Basic.NET code. Paste INSIDE the Format event.

```
Me.SubReport1.Report = rpt
```

To write the code in C#

C# code. Paste INSIDE the Format event.

```
this.subReport1.Report = rpt;
```

Add Code to Layouts Using Script

In a section report, you can use script to access controls, functions in a class, namespaces, etc. You can also create classes inside the script to call methods or add code to a report's script from a Windows Form. The following sections illustrate simple scripting scenarios with examples.

These steps assume that you have already added a Section Report (code based) template in a Visual Studio project. See Adding an ActiveReport to a Project for more information.

To access controls in script

To add script to a report to access a textbox named TextBox1 in the detail section and assign the text "Hello" to it:

1. On the script tab of the report, drop down the **Object** list and select **Detail**. This populates the Event drop-down list with section events.
2. Drop down the **Event** list and select **Format**. This creates script stubs for the event.

To access a textbox in the detail section in VB.NET script

Visual Basic.NET script. Paste INSIDE the Detail Format event.

```
Me.TextBox1.Text = "Hello"
```

Or

Visual Basic.NET script. Paste INSIDE the Detail Format event.

```
CType(rpt.Sections("Detail1").Controls("TextBox1"), TextBox).Text = "Hello"
```

To access a textbox in the detail section in C# script

C# script. Paste INSIDE the Detail Format event.

```
this.textBox1.Text = "Hello";
```

Or

C# script. Paste INSIDE the Detail Format event.

```
((TextBox)rpt.Sections["detail"].Controls["TextBox1"]).Text = "Hello";
```

To give a script access to functions in a class in your project

Using the **AddNamedItem** method, you can allow the script to access functions in a class file within your project. This allows you to keep secure information such as a database connection string or a SQL query string in the code instead of saving it in the RPX file.

1. In the Code View of the Form, add a class to your project named **clsMyItem**.

   To add a class in Visual Basic.NET

   Visual Basic.NET code.

   ```
   Public Class clsMyItem
   End Class
   ```

   To add a class in C#

   C# code.

   ```
   public partial class clsMyItem
   {
   }
   ```

2. Add a public function to your class using code like the following:

   To create a public function in Visual Basic.NET

   Visual Basic.NET code. Paste INSIDE the new class.

   ```
   Public Function getMyItem() As String
       getMyItem = "Hello"
   End Function
   ```

   To create a public function in C#

   C# code. Paste INSIDE the new class.

   ```
   public string getMyItem()
   {
   ```

```
    return "Hello";
}
```

3. Go to the design view of the report and double-click the gray area around the design surface to create an event-handling method for the **ReportStart** event.
4. Add the following code to the handler:

To access the class in Visual Basic.NET

Visual Basic.NET code. Paste before or in the ReportStart event.
`Me.AddNamedItem("myItem", new clsMyItem())`

To access the class in C#

C# code. Paste before or in the ReportStart event.
`this.AddNamedItem("myItem", new clsMyItem());`

5. From the Visual Studio toolbox, drag and drop a TextBox control onto the detail section of the report.
6. Go to the script tab and drop down the **Object** list to select **Detail**. This populates the Event drop-down list with section events.
7. Drop down the **Event** list and select **Format**. This creates script stubs for the event.
8. Add the following script to the event to access a control on the report and populate it using the named item.

To access the control in VB.NET script

VB.NET script. Paste INSIDE the Detail Format event.
`Me.TextBox1.Text = myItem.getMyItem()`

Or

VB.NET script. Paste INSIDE the Detail Format event.
`CType(rpt.Sections("Detail1").Controls("TextBox1"), TextBox).Text = myItem.getMyItem()`

To access the control in C# script

C# script. Paste INSIDE the Detail Format event.
`this.textBox1.Text = myItem.getMyItem();`

Or

C# script. Paste INSIDE the Detail Format event.
`((TextBox)rpt.Sections["detail"].Controls["textBox1"]).Text = myItem.getMyItem();`

9. Go to the preview tab to view the result.

To access namespaces

By using the **AddScriptReference** method, you can gain access to .NET or custom namespaces. This is only necessary if

you need a reference, such as System.Data.dll, that is not initialized in the project before the script runs.

To access a namespace in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the Form code. Replace *YourReportName* with the name of your report.

```
Private Sub runReport()
      Dim rpt as new YourReportName
      rpt.AddScriptReference("System.Data.dll")
      rpt.Run()
End Sub
```

To access a namespace in C#

C# code. Paste INSIDE the Form code. Replace *YourReportName* with the name of your report.

```
private void runReport()
{
      YourReportName rpt = new YourReportName;
      rpt.AddScriptReference("System.Data.dll");
      rpt.Run();
}
```

Note: If you are using the custom assemblies, they must have Strong Name and registered to GAC folder.

If you want to use custom assemblies in the Script section of the Designer, then you need to add the assembly reference to a report before loading it into the report designer. To do this, follow these steps:

1. Create a custom assembly with the strong name and register it to GAC.
2. Create a new sample with a section report (XML-based).
3. Add the report designer to the form.
4. Add the following code to the **Form_Load** event.

   To access a namespace in Visual Basic.NET

   Visual Basic.NET code. Paste INSIDE the Form code. Replace *YourReportName* with the name of your report.

   ```
   Dim rpt As SectionReport = New SectionReport()
   Dim xtr As System.Xml.XmlTextReader = New
   System.Xml.XmlTextReader("..\..\SectionReport2.rpx")
      rpt.LoadLayout(xtr)
      xtr.Close()
      rpt.AddScriptReference("ClassLibrary1.dll")
      designer1.Report = rpt
   ```

   To access a namespace in C#

   C# code. Paste INSIDE the Form code. Replace *YourReportName* with the name of your report.

   ```
   SectionReport rpt = new SectionReport();
   System.Xml.XmlTextReader xtr = new
   System.Xml.XmlTextReader(@"..\..\SectionReport2.rpx");
   ```

```
rpt.LoadLayout(xtr);
xtr.Close();
rpt.AddScriptReference(@"ClassLibrary1.dll");
designer1.Report = rpt;
```

5. Run the sample.
6. Open the **Script** section of the designer.

You can now use the custom assemblies reference in the Script section of the Desginer.

To add code to a report's script from a Windows Form

Using the **AddCode** method in the Code View of the Form, you can add code into the script. The AddCode method allows you to add actual code segments to the script at run time. This is useful for allowing secure information, such as a database connection string or SQL query string, to be used inside the script without saving it in the RPX file.

1. Go to the Code View of your report and add a public function like the following:

   To add code in Visual Basic.NET

   Visual Basic.NET code. Paste INSIDE the report class.

   ```
   Public Function addThisCode() As String
      Dim sCode As String = "Public Function ShowACMessage() As String" +
   Environment.NewLine + "ShowACMessage = ""my Added Code""" + Environment.NewLine +
   "End Function"
      addThisCode = sCode
   End Function
   ```

   To add code in C#

   C# code. Paste INSIDE the report class.

   ```
   public string addThisCode()
   {
      string sCode = "public string ShowACMessage(){return \"my Added Code\";}";
      return sCode;
   }
   ```

2. In the design view of your report double-click the gray area around the design surface to create an event-handling method for the **ReportStart** event.
3. Add the following code to the handler:

   To access the class in Visual Basic.NET

   Visual Basic.NET code. Paste INSIDE the ReportStart event.

   ```
   Me.AddCode(addThisCode())
   ```

   To access the class in C#

   C# code. Paste INSIDE the ReportStart event.

```
this.AddCode(addThisCode());
```

4. Go to the script tab and drop down the **Object** list to select **Detail**. This populates the Event drop-down list with section events.
5. Drop down the Event list and select **Format**. This creates script stubs for the event.
6. Add the following script to the event:

To write the script in Visual Basic.NET

VB.NET script. Paste INSIDE the Detail1_Format event.

```
Me.TextBox1.Text = ShowACMessage()
```

Or

VB.NET script. Paste INSIDE the Detail1_Format event.

```
CType(rpt.Sections("Detail1").Controls("TextBox1"), TextBox).Text = ShowACMessage()
```

To write the script in C#

C# script. Paste INSIDE the detail_Format event.

```
this.textBox1.Text = ShowACMessage();
```

Or

C# script. Paste INSIDE the detail_Format event.

```
((TextBox)rpt.Sections["detail"].Controls["textBox1"]).Text = ShowACMessage();
```

To create classes inside the script to call methods

If the script requires a method to be called, you can construct a class inside the script.

1. Go to the script tab and add the following code at the top:

To create a class inside the script in VB.NET script

VB.NET script. Paste INSIDE the script tab.

```
Public Class MyFuncs
   Public  Sub New()
   End Sub
   Public Function ShowMyString() As String
      Return "This is my string"
   End Function
End Class
```

To create a class inside the script in C#

C# script. Paste INSIDE the script tab.

```
public class MyFuncs
{
   public MyFuncs()
   {
   }
   public string ShowMyString()
   {
      return "This is my string";
   }
}
```

2. On the script tab, now drop down the Object list and select **Detail**. This populates the Event drop-down list with section events.
3. Drop down the Event list and select **Format**. This creates script stubs for the event.
4. Add the following script to the event:

To create a class inside the script in VB.NET script

VB.NET script. Paste INSIDE the Detail1_Format event.

```
Dim f As MyFuncs =  New MyFuncs()
Me.TextBox1.Text = f.ShowMyString
```

Or

VB.NET script. Paste INSIDE the Detail1_Format event.

```
Dim f As MyFuncs =  New MyFuncs()
CType(rpt.Sections("Detail1").Controls("TextBox1"), TextBox).Text = f.ShowMyString
```

To create a class inside the script in C#

C# script. Paste INSIDE the detail_Format event.

```
MyFuncs f = new MyFuncs();
this.textBox1.Text = f.ShowMyString();
```

Or

C# script. Paste INSIDE the detail_Format event.

```
MyFuncs f = new MyFuncs();
((TextBox)rpt.Sections["detail"].Controls["textBox1"]).Text = f.ShowMyString();
```

Note: Use the examples with the "this" (C#) and "Me"(Visual Basic.NET) keywords, as they are recommended rather than the ones with "rpt".

Save and Load RDF Report Files

ActiveReports allows reports to be saved in their own standard format called an RDF file (Report Document Format). In

this format, the data is static. The saved report displays the data that is retrieved when you run the report. You can save a report to an RDF file and load it into the viewer control.

To save a report as a static RDF file

1. Double-click the title bar of the Windows Form to create an event-handling method for the Form_Load event.
2. Add the following code to the handler to save the report.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the Form_Load event.

```
Dim rpt As New YourReportName()
rpt.Run()
rpt.Document.Save(Application.StartupPath + \NewRDF.RDF)
```

To write the code in C#

C# code. Paste INSIDE the Form_Load event.

```
YourReportName rpt = new YourReportName();
rpt.Run();
rpt.Document.Save(Application.StartupPath + \\NewRDF.RDF);
```

To load a saved RDF file into the ActiveReports viewer

1. Double-click the title bar of the Windows Form to create an event-handling method for the Form_Load event.
2. Add the following code to the handler to load the saved report.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the Form_Load event.

```
Viewer1.Document.Load("Location of the .RDF File")
```

To write the code in C#

C# code. Paste INSIDE the Form_Load event.

```
viewer1.Document.Load(@"Location of the .RDF File");
```

Note: The Windows Form Viewer can display RDF files made with any version of ActiveReports, including COM versions.

To save or load report files to a memory stream

1. Double-click the title bar of the Windows Form to create an event-handling method for the Form_Load event.
2. Add the following code to the handler to save the report to a memory stream and load the memory stream into the ActiveReports viewer.

   The following examples show what the code for the method looks like.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the Form_Load event.

```
Dim strm As New System.IO.MemoryStream()
Dim rpt As New YourReportName()
rpt.Run()
rpt.Document.Save(strm)
Dim theBytes(strm.Length) As Byte
strm.Read(theBytes, 0, Int(strm.Length))
strm.Position = 0
Viewer1.Document.Load(strm)
```

To write the code in C#

C# code. Paste INSIDE the Form_Load event.

```
System.IO.MemoryStream strm = new System.IO.MemoryStream();
YourReportName rpt = new YourReportName();
rpt.Run();
rpt.Document.Save(strm);
byte[] theBytes = new byte[strm.Length];
strm.Read(theBytes, 0, (int)strm.Length);
strm.Position =0;
viewer1.Document.Load(strm);
```

Save and Load RPX Report Files

Although ActiveReports writes report layouts in either C# or Visual Basic.NET, you can save the layout of your report as a report XML (RPX) file for portability. If you make changes to the RPX file and load it back into an ActiveReport in Visual Studio, you can see the changes you made reflected in the C# or Visual Basic.NET code in the *YourReportName*.Designer.vb or *YourReportName*.Designer.cs file.

> ⚠ **Caution**: When you load an RPX layout into a report object, it overwrites everything in the report object. In order to avoid overwriting important layouts, add a new blank ActiveReport and load the RPX file onto it.

To save a report as an RPX file at design time

1. From the Visual Studio **Report** menu, select **Save Layout**.
2. In the **Save As** dialog that appears, set the file name and select the location where you want to save it. The file extension is ***.rpx**.
3. Click the **Save** button to save the report layout and close the dialog.

> **Note**: When you save a layout that contains a dataset, ActiveReports saves the data adapter and data connection in the component tray, but not the dataset itself. When the saved layout is loaded into another report, you can regenerate the dataset with the data adapter and data connection.

To load an RPX file at design time

1. From the Visual Studio **Report** menu, select **Load Layout**.
2. In the **Open** dialog that appears, navigate to the location of the .rpx file and select it.
3. Click the **Open** button to load the report layout.

To save a report as an RPX file at run time

Use the **SaveLayout** method to save your report layout at run time.

> **Note**: When you save a report layout, ActiveReports only saves the code in the script editor to the file. Any code behind the report in the .cs or .vb file is not saved to the RPX file.

1. Right-click the Windows Form and select **View Code** to see the code view for the Windows form.
2. Add the following code to the Form class to save the report.

   The following example shows what the code for the method looks like.

   To write the code in Visual Basic.NET

   Visual Basic.NET code. Paste INSIDE the Form class.

   ```
   Dim rpt As New SectionReport1()
   Dim xtw As New System.Xml.XmlTextWriter(Application.StartupPath + "\report.rpx",
   Nothing)
   rpt.SaveLayout(xtw)
   xtw.Close()
   ```

   To write the code in C#

   C# code. Paste INSIDE the Form class.

   ```
   SectionReport1 rpt = new SectionReport1();
   System.Xml.XmlTextWriter xtw = new System.Xml.XmlTextWriter(Application.StartupPath
   + "\\report.rpx", null);
   rpt.SaveLayout(xtw);
   xtw.Close();
   ```

   Save report layouts before they run. If you save a layout after the report runs, you also save any dynamic changes made to properties or sections in the report. To avoid this when you call SaveLayout inside the report code, use the ReportStart event.

   > **Note:** The SaveLayout method uses utf-16 encoding when you save to a stream, and utf-8 encoding when you save to a file.

To load an RPX file into the ActiveReports viewer at run time

1. Right-click on the Windows Form and select **View Code** to see the code view for the Windows form.
2. Add the following code to the form class to load a report.

The following examples show what the code for the method looks like.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the Form class.

```
Dim rpt As New GrapeCity.ActiveReports.SectionReport()
' For the code to work, this report.rpx must be stored in the bin\debug folder of
your project.
Dim xtr As New System.Xml.XmlTextReader(Application.StartupPath + "\report.rpx")
rpt.LoadLayout(xtr)
xtr.Close()
Viewer1.Document = rpt.Document
rpt.Run()
```

To write the code in C#

C# code. Paste INSIDE the Form class.

```
GrapeCity.ActiveReports.SectionReport rpt = new
GrapeCity.ActiveReports.SectionReport();
// For the code to work, this report.rpx must be stored in the bin\debug folder of
your project.
System.Xml.XmlTextReader xtr = new System.Xml.XmlTextReader(Application.StartupPath
+ "\\Sample.rpx");
rpt.LoadLayout(xtr);
xtr.Close();
viewer1.Document = rpt.Document;
rpt.Run();
```

Print Multiple Copies, Duplex and Landscape

In a section report, you can modify various printer settings or print multiple copies of a report at design time and at run time.

Printer Settings

At design time, you can set up duplex printing, page orientation, collation, and page size in the **Printer Settings** tab of the Report Settings Dialog.

To set up duplex printing in Printer Settings

1. In the Report Explorer, double-click the **Settings** node.
2. In the Report Settings dialog that appears, click **Printer Settings**.
3. On the Printer Settings page, next to **Duplex**, select one of the following options:
   - **Printer Default**: The report uses the default settings of the selected printer.
   - **Simplex**: Turns off duplex printing.
   - **Horizontal**: Prints horizontally on both sides of the paper.
   - **Vertical**: Prints vertically on both sides of the paper.

4. Click **OK** to return to the report.

To use code to set up duplex printing

1. Double-click the gray area below the design surface to create an event-handling method for the report's ReportStart event.
2. Add the following code to the handler to set up duplexing.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the ReportStart event.

```
Me.PageSettings.Duplex = System.Drawing.Printing.Duplex.Horizontal
```

To write the code in C#

C# code. Paste INSIDE the ReportStart event.

```
this.PageSettings.Duplex = System.Drawing.Printing.Duplex.Horizontal;
```

To set page orientation on the Printer Settings page

1. In the Report Explorer, double-click the **Settings** node.
2. In the Report Settings dialog that appears, click **Printer Settings**.
3. On the **Printer Settings** page, in the Orientation section, select either **Default**, **Portrait** or **Landscape**.
4. Click **OK** to return to the report.

To use code to change page orientation

1. Double-click the gray area below the design surface to create an event-handling method for the report's ReportStart event.
2. Add the following code to the handler to change the page orientation of the report for printing.

> **Note**: Page orientation can only be modified before the report runs. Otherwise, changes made to the page orientation are not used during printing.

The following example shows what the code for the method looks like.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the ReportStart event.

```
Me.PageSettings.Orientation =
GrapeCity.ActiveReports.Document.Section.PageOrientation.Landscape
```

To write the code in C#

C# code. Paste INSIDE the ReportStart event.

```
this.PageSettings.Orientation =
GrapeCity.ActiveReports.Document.Section.PageOrientation.Landscape;
```

Multiple Copies

You can print multiple copies using the Print dialog in the Preview tab or in the Viewer, or you can use code to set the number of copies to print.

To set multiple copies in the print dialog

1. With a report displayed in the viewer or in the preview tab, click **Print**.
2. In the Print dialog that appears, next to **Number of copies**, set the number of copies that you want to print.

To use code to set multiple copies

1. Double-click in the gray area below the design surface to create an event-handling method for the report's ReportStart event.
2. Add the following code to the handler to set multiple copies of the report for printing.

   The following example shows what the code for the method looks like for printing five copies.

   To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the ReportStart event.
`Me.Document.Printer.PrinterSettings.Copies = 5`
Visual Basic.NET code. Paste INSIDE the ReportEnd event.
`Me.Document.Printer.Print()`

   To write the code in C#

C# code. Paste INSIDE the ReportStart event.
`this.Document.Printer.PrinterSettings.Copies = 5;`
C# code. Paste INSIDE the ReportEnd event.
`this.Document.Printer.Print();`

Localize and Deploy

ActiveReports uses an English locale by default, and includes localization resources for Japanese and Russian locales. You can also localize all of the components into any language you need. GrapeCity may, from time to time and on the agreement of users who localize components, include additional locales with future hot fixes and service packs. If you are willing to share your localized resources with other users, please inform technical support staff so that they can pass on your resource files to development.

There are several ways to deploy your ActiveReports applications. See the topics listed below for more information on customizing, localizing and deploying your applications.

Localize Reports, TextBoxes, and Chart Controls

Learn how to localize individual textboxes, chart controls, and entire reports.

Localize ActiveReports Resources
Learn to localize ActiveReports dialogs, error messages, and images.

Localize the End User Report Designer
Learn to localize UI strings, error messages, and images in Windows Forms Designer control.

Localize the Viewer Control
Learn to localize the strings and images in the Windows Forms Viewer Control.

Deploy Windows Applications
Learn to deploy ActiveReports Windows applications.

Deploy Web Applications
Learn to deploy ActiveReports Windows applications.

Configure HTTPHandlers in IIS 8 and IIS 10
Learn to deploy ActiveReports Windows applications.

Localize Reports, TextBoxes, and Chart Controls

In a section layout report, the Report object, TextBox control, and Chart control have a public Culture property that allows you to localize data when the OutputFormat property is set to D (date), C (currency), or other .NET formats.

> **Note**: The default value for the Culture property is **(default, inherit)**. For the Report object, this is the culture of the current thread and for the TextBox control and the ChartControl, this is the culture of the Report object.

In a page layout report, the Report object, TextBox control, and Chart control all have a Language property that works in the same way. The default value for the Language property is **Default**, which is the culture of the current thread.

Design Time

At design time, you can set the culture or language in the Visual Studio Properties window.

To localize a Report at design time

1. Click the gray area around the design surface to select the Report in the Properties window.
2. In the Properties window, drop down the **Culture** or **Language** property and select the culture that you want to apply to the report.

To localize a TextBox control at design time

1. Click the TextBox control that you want to localize to select it.
2. In the Properties window, drop down the **Culture** or **Language** property and select the culture that you want to apply to the textbox.

To localize a Chart control at design time

1. Click the Chart control to select it.
2. In the Properties window, drop down the **Culture** or **Language** property and select the culture that you want to apply to the chart.

Run Time

You can also specify a culture in code for section reports. For a list of System.Globalization culture codes, see Cultures.

To localize a Report at run time

1. Double-click the gray area around the design surface, to create an event handling method for the ReportStart event.
2. In the code view of the report that appears, paste code like the following.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the ReportStart event.

```
YourReportName.Culture =
System.Globalization.CultureInfo.CreateSpecificCulture("en-US")
```

To write the code in C#

C# code. Paste INSIDE the ReportStart event.

```
YourReportName.Culture =
System.Globalization.CultureInfo.CreateSpecificCulture("en-US");
```

To localize a TextBox at run time

1. On the design surface, double-click the section containing the TextBox control that you want to localize to create an event handling method for the section Format event.
2. In the code view of the report that appears, paste code like the following inside the Format event.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the Format event.

```
TextBox.Culture = System.Globalization.CultureInfo.CreateSpecificCulture("en-US")
```

To write the code in C#

C# code. Paste INSIDE the Format event.

```
textBox.Culture = System.Globalization.CultureInfo.CreateSpecificCulture("en-US");
```

To localize a Chart at run time

1. On the design surface, double-click the section containing the ChartControl that you want to localize to create an event handling method for the section Format event.
2. In the code view of the report that appears, paste code like the following in the Format event.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the Format event.

```
ChartControl.Culture = System.Globalization.CultureInfo.CreateSpecificCulture("en-
US")
```

To write the code in C#

C# code. Paste INSIDE the Format event.

```
chartControl.Culture = System.Globalization.CultureInfo.CreateSpecificCulture("en-US");
```

Localize ActiveReports Resources

You can localize all of the UI strings, error messages, and images that appear in ActiveReports in included resource files, and alter and run a batch file to localize each resource.

To localize the designer control

All of the localization files are located in *C:\Program Files (x86)\GrapeCity\ActiveReports 14\Localization*. The topic illustrates localization taking **End User Report Designer** as an example.

Specify the culture you want to use in the batch files.

1. Start Notepad or another text editor.
2. Open *.bat file for each resource you want to localize and change the **Culture** value to the culture you want to use.
3. Save and close each file.

Localize strings (and images) in the resource files.

1. Copy all the files for the resources you want to localize from the Localization folder: *C:\Program Files (x86)\GrapeCity\ActiveReports 14\Localization*.
2. Paste these files to a local folder. For example, if you want to localize the designer, copy the following files to a local folder, say *D:\Localize*.
   - ARDesigner.zip
   - ARDesigner.bat
   - NameComplete.exe
   - NameComplete.exe.config
   - Localize.bat
   - publickey.snk
3. Extract the zip file of the resource (ARDesigner.zip).
4. In the folder extracted, open each subfolder and change the strings in each of the *.resx files.
   For example, *D:\Localize\ARDesigner\ARDesigner\Res\ARDesigner\Dialogs\DatasourceDialog.resx*

   > **Tip**: Strings are located between <value> and </value> tags in the resource files.
5. If you want to change the images, rename your localized images to the names of the ones in the Res\Resources subfolder and replace them with your localized images.

Run the batch file.

1. From the Start menu, type **cmd** in the text box.
2. Navigate to the local directory by typing **cd D:\Localize** and press Enter.
3. Type the name of the *.bat file and press Enter to run the file. The NameCompleter.exe application runs, and creates the following.
   - A SatelliteAssembly folder inside the resource subfolder.
   - A language subfolder with the name of the culture (say **ja**) you set inside the SatelliteAssemby folder.
   - A localized GrapeCity.ActiveReports.*AssemblyName*.resources.dll file inside the language subfolder.

4. Add verification entry for the assembly by running following command:
   sn.exe -Vr "ja\GrapeCity.ActiveReports.Design.Win.resources.dll".
5. Copy the language subfolder and paste it into the Debug folder of your application.

> **Note**: Before you can distribute or put your localization in the Global Assembly Cache (GAC), you must first send the localized GrapeCity.ActiveReports.*AssemblyName*.resources.dll file to GrapeCity and get it signed with a strong name. Then you can drag the language subfolder with the signed dll file into C:\WINDOWS\ASSEMBLY, or distribute it with your solution.

Test your localized application on a machine that does not share the culture of the localized DLLs.

1. Add the following code in the form's constructor just before the InitializeComponent method is called.
2. Replace the "ja-JP" in the example code with the culture you specified in the *.bat file.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the form's constructor just before the InitializeComponent method.

```
System.Threading.Thread.CurrentThread.CurrentUICulture = New
System.Globalization.CultureInfo("ja-JP")
```

To write the code in C#

C# code. Paste INSIDE the form's constructor just before the InitializeComponent method.

```
System.Threading.Thread.CurrentThread.CurrentUICulture = new
System.Globalization.CultureInfo("ja-JP");
```

Localize the End User Report Designer

You can localize all of the UI strings, error messages, and images that appear in the ActiveReports Windows Forms Designer control in included resource files, and alter and run a batch file to localize the assembly. See topic Localize ActiveReports Resources for localizing the End User Report Designer.

Localize the Viewer Control

You can localize all of the strings and images that appear in the Windows Forms Viewer control in included resource files, and alter and run a batch file to localize the control.

To localize the viewer control

All of the localization files are located in *C:\Program Files (x86)\GrapeCity\ActiveReports 14\Localization*.

Specify the culture you want to use in the batch file.

1. Start Notepad or another text editor.
2. Open the WinViewer.bat file and change the **Culture** value to the culture you want to use.
3. Save and close the file.

Localize strings (and images) in the resource files.

1. Copy the following files required for the WebViewer from the Localization folder: *C:\Program Files (x86)\GrapeCity\ActiveReports 14\Localization:*
    - WinViewer.zip
    - WinViewer.bat
    - NameComplete.exe
    - NameComplete.exe.config
    - Localize.bat
    - publickey.snk
2. Paste these files to a local folder, say *D:\Localize*\.
3. Extract WinViewer.zip file.
4. In the folder extracted, open each subfolder and change the strings in each of the *.resx files.
   For example, *D:\Localize\WinViewer\WinViewer\Res\ActiveReports\Viewer\Win\FlatViewer.resx*
5. If you want to change the images, rename your localized images to the names of the ones in the Res\Resources subfolder and replace them with your localized images.

Run the batch file.

1. From the Start menu, type **cmd** in the text box.
2. Navigate to the local directory by typing **cd D:\Localize** and press Enter.
3. Type **WinViewer.bat** and press Enter to run the file. The NameCompleter.exe application runs, and creates the following.
    - A SatelliteAssembly folder inside the WinViewer folder.
    - A language subfolder with the name of the culture (say **ja**) you set inside the SatelliteAssemby folder.
    - A localized GrapeCity.ActiveReports.Viewer.Win.resources.dll file inside the language subfolder.
4. Add verification entry for the assembly by running following command:
   sn.exe -Vr "ja\GrapeCity.ActiveReports.Design.Win.resources.dll".
5. Copy the language subfolder and paste it into the Debug folder of your application.

> **Note**: Before you can distribute or put your localization in the Global Assembly Cache (GAC), you must first send the localized GrapeCity.ActiveReports.Viewer.Win.resources.dll file to GrapeCity and get it signed with a strong name. Then you can drag the language subfolder with the signed dll file into C:\WINDOWS\ASSEMBLY.

Test your localized application on a machine that does not share the culture of the localized DLL.

1. Add the following code in the form's constructor just before the InitializeComponent method is called.
2. Replace the "ja" in the example code with the culture you specified in the WinViewer.bat file.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the form's constructor just before the InitializeComponent method.

```
System.Threading.Thread.CurrentThread.CurrentUICulture = New
System.Globalization.CultureInfo("ja")
```

To write the code in C#

C# code. Paste INSIDE the form's constructor just before the InitializeComponent method.

```
System.Threading.Thread.CurrentThread.CurrentUICulture = new
System.Globalization.CultureInfo("ja");
```

Deploy Windows Applications

Before you begin

Before deploying a Windows application, there are a few settings that can be helpful.

- On report files, in the Properties window, change the **Copy to Output Directory** property to **Copy always**.
- For ActiveReports references, change the **Copy Local** property to **True**. This puts all of the needed reference assemblies into the Release folder when you build your project.

It is also good to be sure that all of the references you need for your reports are included. Here is a table listing features and required DLLs.

Features and References

These assemblies are added automatically when you add controls to forms or report controls to code-based section reports, but Visual Studio does not do this with XML-based (RPX and RDLX) reports.

Feature	Required Assembly
Export: Excel, Html, Image/Tiff, Pdf, Word/Rtf, Text/Xml	GrapeCity.ActiveReports.Export.*.dll (replace * with Excel, Html, Image, Pdf, Word, or Xml)
Viewer control (or printing without the Viewer control)	GrapeCity.ActiveReports.Viewer.Win.dll
Designer, Toolbox, or ReportExplorer control	GrapeCity.ActiveReports.Design.Win.dll

XCopy Deployment

1. Open your project in Visual Studio, and set the **Solution Configuration** to Release.
2. From the Build menu, select **Build Solution**.
3. In Windows Explorer, navigate to the project's bin directory, and copy the files from the **Release** folder into a zip file.
4. Distribute the zip file.

MSI Installer Deployment

To create an installer project

1. Open an existing ActiveReports project or create a new one.
2. From the Visual Studio **Build** menu, select **Build *YourActiveReportsProjectName*** to build your report project.
3. From the **File** menu, select **Add**, then **New Project** to open the **Add New Project** dialog.
4. In the Add New Project dialog under Project Types, expand the **Other Project Types** node and select **Setup and Deployment**.
5. Under Visual Studio Installer, select **Setup Project,** rename the file and click **OK**. The **ProductName** that you enter determines the name that is displayed for the application in folder names and in the **Programs and Features** control panel item.
6. In the File System editor that appears, under File System on Target Machine, select the **Application Folder**.

> **Note**: To show the File System editor at any time, drop down the **View** menu and select **Editor**, then **File System**.

7. From the Visual Studio **Action** menu, select **Add**, then **Project Output**.
8. In the **Add Project Output Group** dialog that appears, choose your ActiveReports project name from the drop-down list.
9. In the list, select **Primary Output** and click **OK**. This adds all of the existing assembly dependencies to your project.
10. If you want to add other ActiveReports DLLs to the installer (e.g. if you use OleObjects on reports, you need to include the Interop.dll), in the Solution Explorer, right-click the installer project name, select **Add**, then **Assembly**.

    > **Note**: If you would rather use the ActiveReports .msm file, please contact our technical support team.

11. In the Select Component dialog that appears, select any components that you want to add and click the **OK** button.
12. From the Visual Studio **Build** menu, select **Build *YourInstallerProjectName*** to build your Installer project.

To deploy the installer application

1. Select the Installer project in the Solution Explorer.
2. From the Visual Studio **Project** menu, click **Install**.
3. The Installer application runs and installs the project on your computer. The distributable exe and msi setup files appear in your installer project Debug folder.

Deploy Web Applications

Follow this guide to deploy ActiveReports Web projects to your Web server. For Web projects using the Professional Edition HttpHandlers, see Configure HTTPHandlers in IIS 8 and IIS 10.

Before you begin

To deploy ActiveReports Web projects, you must have access to the Microsoft .NET Framework version 4.6.2 or higher and the coordinating version of ASP.NET, or ASP.NET Core with .NET Core 2.1 and above. You must also have access to Internet Information Services version 8 or higher, and you need administrative access to the server.

It is also good to be sure that all of the references you need for your reports are included. Here is a table listing features and required DLLs.

Features and References

These assemblies are added automatically when you add controls to forms or report controls to code-based section reports, but Visual Studio does not do this with XML-based (RPX and RDLX) reports.

Feature	Required Package
Export: Excel	GrapeCity.ActiveReports.Export.Excel GrapeCity.ActiveReports.Core.Export.Excel.Page
Export: HTML	GrapeCity.ActiveReports.Export.Html GrapeCity.ActiveReports.Core.Export.Html.Page
Export: Image	GrapeCity.ActiveReports.Export.Image GrapeCity.ActiveReports.Core.Export.Image.Page
Export: PDF	GrapeCity.ActiveReports.Export.Pdf GrapeCity.ActiveReports.Core.Export.Pdf.Page
Export: Word/RTF	GrapeCity.ActiveReports.Export.Word.dll

Export: XML	GrapeCity.ActiveReports.Export.Xml
Export: CSV/JSON	GrapeCity.ActiveReports.Core.Export.Text.Page
WebViewer or HttpHandlers (Pro Edition only)	GrapeCity.ActiveReports.Web Grapecity.ActiveReports.Web.Viewer
JSViewer	GrapeCity.ActiveReports.Aspnetcore.Viewer GrapeCity.ActiveReports.Aspnet.Viewer
Web Designer	GrapeCity.ActiveReports.Aspnetcore.Designer GrapeCity.ActiveReports.Aspnet.Designer

Other packages that are required for deploying web applications are as follows:

- GrapeCity.ActiveReports
- GrapeCity.ActiveReports.Core.Rdl
- Newtonsoft.Json
- Gcef.Data.DataEngine
- Gcef.Data.VBFunctionLib
- Gcef.Data.ExpressionInfo
- Grapecity.DataVisualization
- GrapeCity.ActiveReports.Core.Rendering
- GrapeCity.ActiveReports.Core.Drawing.Gdi
- GrapeCity.ActiveReports.Core.DataProviders
- GrapeCity.ActiveReports.Chart
- GrapeCity.ActiveReports.Document

To install prerequisites on the server

Follow Microsoft's instructions to install each of the following on your Web server:

- The Microsoft .NET Framework version 4.6.2 or higher
- ASP.NET version 4.6.2 or higher (must be the same version as the Framework)
- ASP.NET Core with .NET Core 2.1 and above.
- Internet Information Services (IIS) version 8

To map your application to a virtual directory

1. Make sure your project is deployed to a virtual directory in IIS.
2. To map requests from the page with the WebViewer control to your virtual directory, paste the **<base>** tag with a virtual directory name inside the **<head>** tag of the page containing the WebViewer control. In this case, all requests to backend assemblies will include the specified virtual directory name. For example,

   ```
   <head>
    <base href="/VirtDirName/">
   </head>
   ```

To set permissions on the server

Depending on your project, you may need to set permissions to allow ActiveReports access to data or folders.

Some examples of required permissions on the server:

- If you are saving files (e.g. PDF or RDF exports) to a folder on Windows machines, the **ASPNET** user ID needs **Write** access to that folder.
- Windows is user configurable, so use **the name assigned to the ASPNET user** instead.
- If your application reads anything from any folder, assign **Read** access to it.
- If your reports run on any networked data source (e.g. SQL, Access, etc.) assign **Read** access to it.
- If you use CacheToDisk, assign **IsolatedStorageFilePermission** to it.

Configure HTTPHandlers in IIS 8 and IIS 10

HttpHandlers are included in the Professional edition of ActiveReports to allow you to quickly and easily display reports in the browser.

Follow these steps to configure the ActiveReports HTTP handlers in IIS so that you can link directly to reports in your Web applications. Once the handlers are configured, you can automatically run a report and view it in the browser from a URL.

> **Note:** The WebViewer, the JSViewer, and the Web Designer are supported only in the Integrated pipeline mode. You will get PlatformNotSupportedException on using these in Classic pipeline mode.

Classic Mode

If any part of your Web application is not supported in Integrated Mode, you can run it using the Classic .NET AppPool.

To run your Web application in the Classic .NET Application Pool

1. In the Control Panel, open **Administrative Tools**, then **Internet Information Services (IIS) Manager**.
2. In the Internet Information Services (IIS) Manager window that appears, in the left pane under Connections, expand the **Sites** node and select the Web application you want to configure.
3. To the right of the Handler Mappings pane that appears, under **Actions**, click **Basic Settings**.
4. In the Edit Site dialog that appears, click the **Select** button.
5. In the Select Application Pool dialog that appears, drop down the Application pool, select **Classic .NET AppPool**, and click **OK**.
6. Back in the Edit Site dialog, click **OK** to accept the changes.

To configure ActiveReports HTTP handlers to enable report linking in your Web applications

1. In the Control Panel, open **Administrative Tools**, then **Internet Information Services (IIS) Manager**.
2. In the Internet Information Services (IIS) Manager window that appears, in the left pane under Connections, expand the **Sites** node and select the Web application you want to configure.
3. In the site's Home pane that appears, under IIS, double-click **Handler Mappings**.
4. To the right of the Handler Mappings pane that appears, under **Actions**, click **Add Script Map**.
5. In the **Add Script Map** dialog that appears, enter the information from the first row of the table below.

   > **Note:** If you have a 64 bit app pool, add script mappings for the 64 bit version of the aspnet_isapi.dll by navigating to C:\Windows\Microsoft.NET\Framework64\v*\aspnet_isapi.dll.

Request path	Executable	Name
*.Web	aspnet_isapi.dll version to match your app pool	ActiveReports 14 Cache Item Script Mapping
*.rpx	aspnet_isapi.dll version to match your app pool	ActiveReports 14 RPX Script Mapping
*.rdlx	aspnet_isapi.dll version to match your app pool	ActiveReports 14 RDLX Script Mapping

*.rdl	aspnet_isapi.dll version to match your app pool	ActiveReports 14 RDL Script Mapping

6. Click the **Request Restrictions** button and ensure that the **Invoke handler only if request is mapped to** check box is cleared.
7. Click **OK** to close the window and add the script mapping.
8. Repeat for each script mapping in the table above.

To add handlers without configuring IIS 8 and IIS 10 using the Classic .NET AppPool

1. In your Web application, open the Web.config file and add code like the following between the <system.web> and </system.web> tags, changing the **ActiveReports Version** number on each line to match the version installed on your machine.

Paste inside the <system.web> tags.

```
<httpHandlers>
    <add verb="*" path="*.rpx"
type="GrapeCity.ActiveReports.Web.Handlers.RpxHandler, GrapeCity.ActiveReports.Web,
Version=14.x.xxxx.0, Culture=neutral, PublicKeyToken=cc4967777c49a3ff" />
    <add verb="*" path="*.rdl"
type="GrapeCity.ActiveReports.Web.Handlers.RdlxHandler,
GrapeCity.ActiveReports.Web, Version=14.x.xxxx.0, Culture=neutral,
PublicKeyToken=cc4967777c49a3ff" />
    <add verb="*" path="*.rdlx"
type="GrapeCity.ActiveReports.Web.Handlers.RdlxHandler,
GrapeCity.ActiveReports.Web, Version=14.x.xxxx.0, Culture=neutral,
PublicKeyToken=cc4967777c49a3ff" />
    <add verb="*" path="*.Web"
type="GrapeCity.ActiveReports.Web.Handlers.WebCacheAccessHandler,
GrapeCity.ActiveReports.Web, Version=14.x.xxxx.0, Culture=neutral,
PublicKeyToken=cc4967777c49a3ff" />
</httpHandlers>
```

2. In your Web application, open the Web.config file and add code like the following between the <system.webServer> and </system.webServer> tags depending on the .Net Framework version installed on your machine.

.Net 4.x

Paste inside the <system.webServer> tags.

```
<handlers>
      <add name="AR14Rpx" path="*.rpx" verb="*" modules="IsapiModule"
scriptProcessor="%windir%\Microsoft.NET\Framework\v4.x\aspnet_isapi.dll"
preCondition="classicMode,runtimeVersionv4.x,bitness32"/>
      <add name="AR14Rdlx" path="*.rdlx" verb="*" modules="IsapiModule"
scriptProcessor="%windir%\Microsoft.NET\Framework\v4.x\aspnet_isapi.dll"
preCondition="classicMode,runtimeVersionv4.x,bitness32"/>
      <add name="AR14Rdl" path="*.rdl" verb="*" modules="IsapiModule"
scriptProcessor="%windir%\Microsoft.NET\Framework\v4.x\aspnet_isapi.dll"
```

```
preCondition="classicMode,runtimeVersionv4.x,bitness32"/>
      <add name="AR14Web" path="*.Web" verb="*" modules="IsapiModule"
scriptProcessor="%windir%\Microsoft.NET\Framework\v4.x\aspnet_isapi.dll"
preCondition="classicMode,runtimeVersionv4.x,bitness32"/>
</handlers>
```

Note: If you have a 64 bit Web application, change the **preCondition** attribute on each line to **classicMode,runtimeVersionv4.x,bitness64**.

Integrated Mode

To configure ActiveReports HTTP handlers to enable report linking in your Web applications

1. In the Control Panel, open **Administrative Tools**, then **Internet Information Services (IIS) Manager**.
2. In the Internet Information Services (IIS) Manager window that appears, in the left pane under Connections, expand the **Sites** node and select the Web application you want to configure.
3. In the site's Home pane that appears, under IIS, double-click **Handler Mappings**.
4. To the right of the Handler Mappings pane that appears, under **Actions**, click **Add Managed Handler**.
5. In the Add Managed Handler dialog that appears, enter the information from the first row of the table below.

Request path	Type	Name
*.Web	GrapeCity.ActiveReports.Web.Handlers.WebCacheAccessHandler	ActiveReports 14 cache item integrated handler mapping
*.rpx	GrapeCity.ActiveReports.Web.Handlers.RpxHandler	ActiveReports 14 RPX integrated handler mapping
*.rdlx	GrapeCity.ActiveReports.Web.Handlers.RdlxHandler	ActiveReports 14 RDLX integrated handler mapping
*.rdl	GrapeCity.ActiveReports.Web.Handlers.RdlxHandler	ActiveReports 14 RDL integrated handler mapping

6. Click the **Request Restrictions** button and ensure that the **Invoke handler only if request is mapped to** check box is cleared.
7. Click **OK** to close the window and add the handler mapping.
8. Repeat for each handler mapping in the table above.

To add handlers without configuring IIS 8 or IIS 10 using the DefaultAppPool

In your Web application, open the Web.config file and add code like the following between the <system.webServer> and </system.webServer> tags, changing the **ActiveReports Version** number on each line to match the version installed on your machine.

Paste inside the <system.webServer> tags.

```
<add verb="*" path="*.Web"
type="GrapeCity.ActiveReports.Web.Handlers.WebCacheAccessHandler,
GrapeCity.ActiveReports.Web, Version=14.x.xxxx.0, Culture=neutral,
PublicKeyToken=cc4967777c49a3ff" name="AR14_WebCacheAccessHandler"
resourceType="Unspecified" preCondition="integratedMode"/>
<add verb="*" path="*.rpx" type="GrapeCity.ActiveReports.Web.Handlers.RpxHandler,
```

```
GrapeCity.ActiveReports.Web, Version=14.x.xxxx.0, Culture=neutral,
PublicKeyToken=cc4967777c49a3ff" name="AR14_RpxHandler" resourceType="Unspecified"
preCondition="integratedMode"/>
<add verb="*" path="*.rdl,*.rdlx"
type="GrapeCity.ActiveReports.Web.Handlers.RdlxHandler, GrapeCity.ActiveReports.Web,
Version=14.x.xxxx.0, Culture=neutral, PublicKeyToken=cc4967777c49a3ff"
name="AR14_RdlxHandler" resourceType="Unspecified" preCondition="integratedMode"/>
```

Note: If you have a 64 bit Web application, change the **preCondition** attribute on each line to **integratedMode,runtimeVersionv4.x,bitness64**.

Samples and Walkthroughs

To understand some of the more complex tasks you can accomplish using ActiveReports, you can open included sample projects, or you can follow walkthroughs, step-by-step tutorials that walk you through every step required to create a specific type of report.

Topic	Content
Samples	Browse brief descriptions of included samples, and follow links that open sample projects in Visual Studio.
Walkthroughs	Look through tutorials that teach you all of the steps involved in creating various types of ActiveReports projects, from the basic report through more complex unbound reports and Web options.

Samples

The ActiveReports 14 samples are available on GitHub. These samples are categorized under three folders - Samples, OnlineSamples, and WebSamples, and further based on features.

Samples

Sample		Description
Advanced	**Page and RDL Reports**	
	Calendar	This sample demonstrates using Calendar data region in reports.
	Custom Chart	This sample demonstrates using custom report item - Radar chart in a report.
	Custom data Provider	This sample demonstrates how to create a project using custom data provider and how to pull data from a comma separated value (CSV) file.
	Custom Pdf Export	This sample demonstrates exporting reports to PDF format using third-party assemblies.
	Custom Resource Locator	This sample showcases a custom implementation of the resource locator to load pictures from the user's "My Pictures" directory.
	Custom Tile Provider	This sample demonstrates how to create a custom tile provider.

	Svg Image	This sample illustrates using SVG as image format in ActiveReports.
	RTF Control	This sample illustrates using RTF control in Page/RDL reports.
	Oracle Data Provider	This sample illustrates using Oracle Data Provider as data source for designing Page/RDL reports.
	Section Reports	
	Custom Drill Through	This sample demonstrates using hyperlinks and the viewer hyperlink event to simulate drill-down from one report to another.
	Custom Word Export	This sample demonstrates exporting Section report to Word format using third-party assemblies.
API	**Page and RDL Reports**	
	Create Report	This sample demonstrates how to create a page report layout in code. It further shows creating a table control, adding table rows and table columns inside it, adding cells inside the table rows and columns and adding text boxes inside the cells.
	Digital Signature Pro	This sample demonstrates how to add digital signatures when exporting to PDF format.
	Export	This sample demonstrates how to export Page and RDL reports to different export formats.
	Layers	This sample demonstrates how to use Layers in a report.
	Report Wizard	This sample demonstrates how to create a custom Report Wizard that allows you to select a report from the list of multiple reports and then allows you to select the data that you want to display in the selected report.
	Stylesheets	This sample demonstrates how to work with embedded and external style sheets in Page and RDL reports.
	Section Reports	
	Charting	This sample demonstrates chart types used in different scenarios, in both bound and unbound modes.
	Cross Section Controls	This sample demonstrates the use of the cross section lines and boxes.
	Cross Tab Report	This sample demonstrates using unbound data, conditional highlighting and distributing data across columns to create a cross-tab view and data aggregation.
	Custom Annotation	This sample demonstrates adding the Custom Annotation button to the report Viewer toolbar and adding a new annotation to the report.
	Digital Signature Pro	This sample demonstrates adding the Custom Annotation button to the report Viewer toolbar and adding a new annotation to the report.
	Export	This sample demonstrates how to export to different export formats using code.
	Inheritance	This sample demonstrates using the method that inherits a report at run time

		and design time.
	Print Multiple Pages per Sheet	This sample demonstrates printing a document with multiple pages per sheet by using the common PrintDocument class of the NET.Framework.
	Style Sheets	This sample demonstrates changing styles at run time to provide a different look to a same report.
	Sub Report	This sample demonstrates using subreports in an ActiveReports report.
	Summary	This sample demonstrates how to display summarized data in a section report.
Data Binding	**Page and RDL Reports**	
	CSV Data Source	This sample demonstrates how to connect to a CSV data source.
	DataSet DataSource	This sample demonstrates how to use a dataset as a data source for a report.
	Json Data Source	This sample demonstrates how to use the Json data provider at run time and add a web service for authentication.
	Object Data Source	This sample demonstrates how to use Object provider for binding a report.
	OData Data Source	This sample demonstrates how to use OData EndPoint for binding a report.
	OleDb Data Source	This sample demonstrates how to connect to an OleDb data source at run time and pass data to the report using LocateDataSource event.
	Xml Data Source	This sample demonstrates how to connect to a XML data source at run time and pass data to the report using LocateDataSource event.
	Section Reports	
	Bound Data	This sample demonstrates binding to ADO.NET Data objects.
	IList Binding	This sample demonstrates creating a custom collection that stores data from the database in the List. The custom collection is displayed by binding data to the DataGridView control by using the DataSource property of this control.
	LINQ	This sample demonstrates how to use LINQ in an ActiveReports report.
	Unbound Data	This sample demonstrates how to create a dataset for a section report and use the FetchData event to populate the Fields collection to display the report unbound data.
	XML	This sample demonstrates how to create a report with XML data, using a SubReport or using the XML hierarchical structure.
Designer Pro	Map	This sample demonstrates how to work with Map control in ActiveReports.
	End User Designer	This sample demonstrates a custom end-user report designer that can be integrated in your applications to allow users to modify report layouts.
	Reports Gallery	This sample demonstrates customizing End User Designer application to display a list of categorized reports.
	Table of Contents	This sample demonstrates how to use TableofContents control in ActiveReports.

Desktop	WPF Viewer	This sample demonstrates using WPF Viewer in a WPF application.
	Win Viewer	This sample demonstrates using Win Viewer in a Windows Form application.
Web	Custom Preview	The sample demonstrates exporting an ActiveReports report to the HTML or PDF format in your Web application.

Web Samples

Sample	Description
JSViewer Angular(Core)	This sample demonstrates the use of the GrapeCity ActiveReports JSViewer with an Angular 8 app and ASP.NET Core back end.
JSViewer MVC	This sample demonstrates the use of GrapeCity ActiveReports JSViewer with an ASP.NET MVC 5 back end.
JSViewer MVC(Core)	This sample demonstrates the use of GrapeCity ActiveReports JSViewer with an ASP.NET MVC Core back end.
JSViewer React(Core)	This sample demonstrates the use of GrapeCity ActiveReports JSViewer with an ReactJS app and ASP.NET Core back end.
JSViewer Vue(Core)	This sample demonstrates the use of GrapeCity ActiveReports JSViewer with an VueJS app and ASP.NET Core back end.
JSViewer Blazor	The sample demonstrates the use of the GrapeCity ActiveReports JSViewer and the Blazor Framework that allows building client-side web applications with C#.
Web Designer MVC	This sample demonstrates Web Designer with an ASP.NET MVC 5 back end.
Web Designer MVC(Core)	This sample demonstrates Web Designer with an ASP.NET MVC Core back end.
Web Designer Angular(Core)	This sample demonstrates Web Designer with an Angular 8 app and ASP.NET Core back end.
WebViewer Pro	This Active Reports Web Pro Sample demonstrates the use of Professional Edition ASP.NET features, such as HTTP handlers, parameterized reports, and more.

Online Samples

Sample	Description
Financial Portfolio	This sample demonstrates visualizing stock data with a Candlestick Chart.
Plant Performance	This sample demonstrates how to visualize the Plant Performance KPIs using the Map control.
ReportsGallery_Angular	This sample demonstrates a variety of RDL, Page, and Section reports along with their descriptions.

Samples

The samples in Samples14 folder demonstrate report designing features that cover desktop, designer, API as well as advanced features. Download these samples from following link:

https://github.com/activereports/Samples14

> **Note**: When you preview a sample report in the **Preview** tab of the ActiveReports Designer in Visual Studio, or in a Standalone Designer, you will get the "*.mdb is not a valid path" error. This error appears because the report data source uses the relative path from the **Bin>Debug** folder of the Sample.

Each sample has a **C#** and a **Visual Basic.NET** code example for Visual Studio. You can also see the comments within the sample projects throughout code.

Sample		Description
Advanced	**Page and RDL Reports**	
	Calendar	This sample demonstrates using Calendar data region in reports.
	Custom Chart	This sample demonstrates using custom report item - Radar chart in a report.
	Custom data Provider	This sample demonstrates how to create a project using custom data provider and how to pull data from a comma separated value (CSV) file.
	Custom Pdf Export	This sample demonstrates exporting reports to PDF format using third-party assemblies.
	Custom Resource Locator	This sample showcases a custom implementation of the resource locator to load pictures from the user's "My Pictures" directory.
	Custom Tile Provider	This sample demonstrates how to create a custom tile provider.
	Svg Image	This sample illustrates using SVG as image format in ActiveReports.
	RTF Control	This sample illustrates using RTF control in Page/RDL reports.
	Oracle Data Provider	This sample illustrates using Oracle Data Provider as data source for designing Page/RDL reports.
	Section Reports	
	Custom Drill Through	This sample demonstrates using hyperlinks and the viewer hyperlink event to simulate drill-down from one report to another.
	Custom Word Export	This sample demonstrates exporting Section report to Word format using third-party assemblies.
API	**Page and RDL Reports**	
	Create Report	This sample demonstrates how to create a page report layout in code. It further shows creating a table control, adding table rows and table columns inside it, adding cells inside the table rows and columns and adding text boxes inside the cells.
	Digital Signature Pro	This sample demonstrates how to add digital signatures when exporting to

		PDF format.
	Export	This sample demonstrates how to export Page and RDL reports to different export formats.
	Layers	This sample demonstrates how to use Layers in a report.
	Report Wizard	This sample demonstrates how to create a custom Report Wizard that allows you to select a report from the list of multiple reports and then allows you to select the data that you want to display in the selected report.
	Stylesheets	This sample demonstrates how to work with embedded and external style sheets in Page and RDL reports.
	Section Reports	
	Charting	This sample demonstrates chart types used in different scenarios, in both bound and unbound modes.
	Cross Section Controls	This sample demonstrates the use of the cross section lines and boxes.
	Cross Tab Report	This sample demonstrates using unbound data, conditional highlighting and distributing data across columns to create a cross-tab view and data aggregation.
	Custom Annotation	This sample demonstrates adding the Custom Annotation button to the report Viewer toolbar and adding a new annotation to the report.
	Digital Signature Pro	This sample demonstrates how to add digital signatures when exporting to PDF format.
	Export	This sample demonstrates how to export to different export formats using code.
	Inheritance	This sample demonstrates using the method that inherits a report at run time and design time.
	Print Multiple Pages per Sheet	This sample demonstrates printing a document with multiple pages per sheet by using the common PrintDocument class of the NET.Framework.
	Style Sheets	This sample demonstrates changing styles at run time to provide a different look to a same report.
	Sub Report	This sample demonstrates using subreports in an ActiveReports report.
	Summary	This sample demonstrates how to display summarized data in a section report.
Data Binding	**Page and RDL Reports**	
	CSV Data Source	This sample demonstrates how to connect to a CSV data source.
	DataSet DataSource	This sample demonstrates how to use a dataset as a data source for a report.
	Json Data Source	This sample demonstrates how to use the Json data provider at run time and add a web service for authentication.
	Object Data Source	This sample demonstrates how to use Object provider for binding a report.

	OData Data Source	This sample demonstrates how to use OData EndPoint for binding a report.
	OleDb Data Source	This sample demonstrates how to connect to an OleDb data source at run time and pass data to the report using LocateDataSource event.
	Xml Data Source	This sample demonstrates how to connect to a XML data source at run time and pass data to the report using LocateDataSource event.
	Section Reports	
	Bound Data	This sample demonstrates binding to ADO.NET Data objects.
	IList Binding	This sample demonstrates creating a custom collection that stores data from the database in the List. The custom collection is displayed by binding data to the DataGridView control by using the DataSource property of this control.
	LINQ	This sample demonstrates how to use LINQ in an ActiveReports report.
	Unbound Data	This sample demonstrates how to create a dataset for a section report and use the FetchData event to populate the Fields collection to display the report unbound data.
	XML	This sample demonstrates how to create a report with XML data, using a SubReport or using the XML hierarchical structure.
Designer Pro	Map	This sample demonstrates how to work with Map control in ActiveReports.
	End User Designer	This sample demonstrates a custom end-user report designer that can be integrated in your applications to allow users to modify report layouts.
	Reports Gallery	This sample demonstrates customizing End User Designer application to display a list of categorized reports.
	Table of Contents	This sample demonstrates how to use TableofContents control in ActiveReports.
Desktop	WPF Viewer	This sample demonstrates using WPF Viewer in a WPF application.
	Win Viewer	This sample demonstrates using Win Viewer in a Windows Form application.
Web	Custom Preview	The sample demonstrates exporting an ActiveReports report to the HTML or PDF format in your Web application.

Advanced

The samples in the Advanced folder describe advanced features separately for:

- Page and RDL Reports
- Section Reports

Page and RDL Reports

This section discusses following samples describing various features in Page and Rdl reports:

Calendar
: This sample demonstrates using Calendar data region in reports.

Custom Chart
: This sample demonstrates using custom report item - Radar chart in a report.

Custom Data Provider
: This sample demonstrates how to create a project using custom data provider and how to pull data from a comma separated value (CSV) file.

Custom Pdf Export
: This sample demonstrates exporting reports to PDF format using third-party assemblies.

Custom Resource Locator
: This sample showcases a custom implementation of the resource locator to load pictures from the user's "My Pictures" directory.

Custom Tile Provider
: This sample demonstrates how to create a custom tile provider.

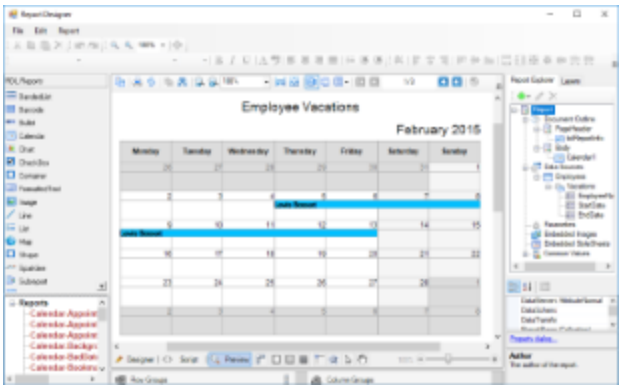
Svg Image
: This sample illustrates using SVG as image format in ActiveReports.

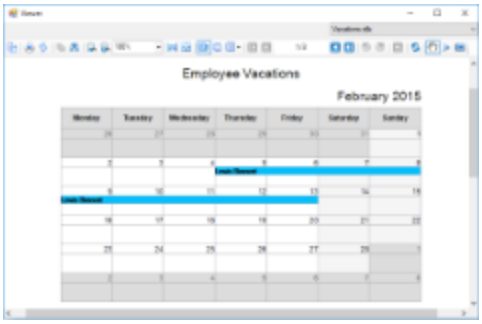
RTF Control
: This sample illustrates using RTF control in Page/RDL reports.

Oracle Data Provider
: This sample illustrates using Oracle Data Provider as data source for designing Page/RDL reports.

Calendar

The Calendar control is now moved to samples to allow customers to continue using this data region for displaying date-based data or events in a calendar format.





Sample Location

..\Samples14\Advanced\PageAndRDL\Calendar

Details

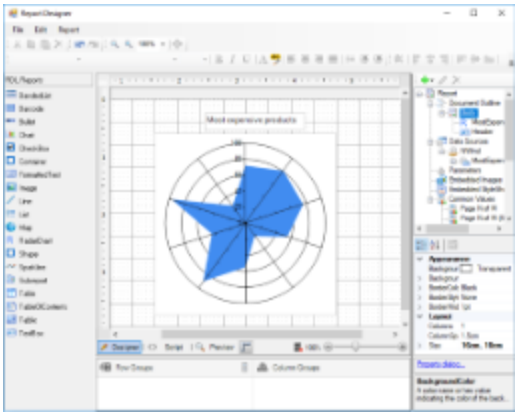
This sample consists of following projects; all data and files are taken from the Calendar data region itself:

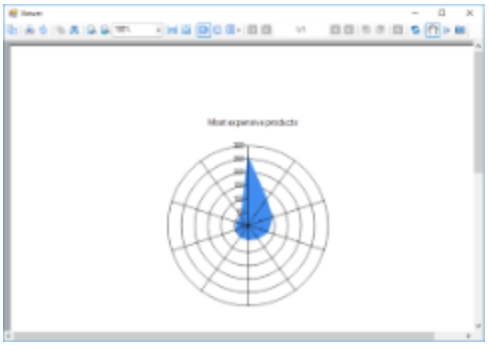
- CalendarComponent: It implements **ICustomReportItem** interface to render Calendar, **IDataRegion** interface for data binding, and **IImageRenderer** interface to render calendar content range to the canvas.
- CalendarDesigner: The designer is inherited from **CustomReportItemDesigner** class. It implements property initialization, glyph drawing, and evaluation utils.
- TestDesignerPro: This is the default start up project. On running this project, an RDL report with a calendar is displayed on the designer. You can also drag and drop and use the Calendar data region available on the toolbox.
- TestViewer: On running this project, a calendar is rendered on Windows Forms Viewer.
- Tests: It contains code for proper functioning of the sample.

Customers who are required to use calendar should now compile and distribute **CalendarComponent** and **CalendarDesigner** assemblies. Binding should be done through GrapeCity.ActiveReports.config file (see test applications).

Custom Chart

This sample illustrates creating custom report item - Radar Chart. The **ICustomReportItem** interface is used to implement custom control, which is radar chart. The designer inherited from **CustomReportItemDesigner** class allows the chart to be available on the designer. The sample uses shared data source Nwind.rdsx.





Sample Location

Visual Basic.NET

```
..\Samples14\Advanced\PageAndRDL\CustomChart\VB.NET
```

C#

```
..\Samples14\Advanced\PageAndRDL\CustomChart\C#
```

Details

When you run this sample, an RDL report 'Radar.rdlx' with Radar chart is displayed on the designer. Go to the Preview tab of the designer to view the report with data pulled from Nwind.rdsx.

The sample consists of following projects:

- RadarChart: It implements **ICustomReportItem** interface to render Radar chart and **IDataRegion** for data binding. The **IImageRenderer** interface renders data to image; this renderer accesses custom data grouping (note that series property name should be same as that defined in designer) and reads values for the chart series.

- RadarDesigner: The designer is inherited from **CustomReportItemDesigner** class. To render one series, one data grouping is added in the **Initialize** method. Custom properties called DataSetName and SeriesValue are added which can be changed in the designer, see classes **DataSetNamesConverter** and **RadarValuesConverter**. For design-time rendering, **RadarControlGlyph** class implements overriding **ControlGlyph** property of the designer and rendering stub data. In this glyph, the **MovableBehavior** method implements moving and resizing of the chart control.

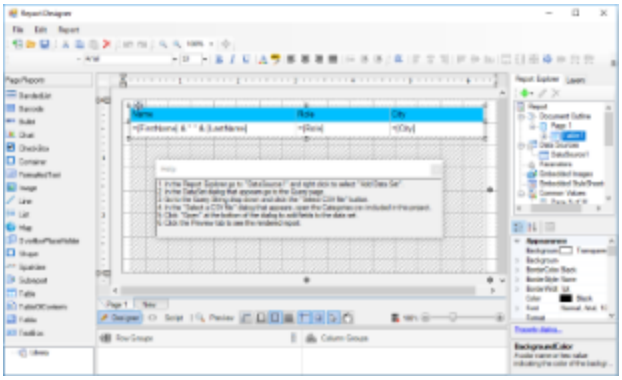
- TestDesignerPro: This is the default start up project. On running this project, an RDL report with Radar chart is displayed on the designer. You can change the chart properties from the Properties pane. You can also drag and drop and use the Radar Chart control available on the toolbox.

- TestViewer: On running this project, Radar chart is rendered on Windows Forms Viewer.

The custom report item (**RadarChart**) and its designer (**RadarDesigner**) are defined in **GrapeCity.ActiveReports.config** file placed in test application projects.

Custom Data Provider

The Custom Data Provider sample demonstrates how to create a project that use a custom data provider and how to pull data from a comma separated value (CSV) file. This sample is part of the ActiveReports Professional Edition.



Sample Location

Visual Basic.NET

```
..\Samples14\Advanced\PageAndRDL\CustomDataProvider\VB.NET
```

C#

```
..\Samples14\Advanced\PageAndRDL\CustomDataProvider\C#
```

Details

When you run this sample, a **DesignerForm** displaying the **DemoReport.rdlx** report in ActiveReports Designer and a **HelperForm** explaining the steps to connect a report to the comma separated value (CSV) file appears.

In the ActiveReports Designer, you can add a dataset with a comma separated values (CSV) file. For adding this file,in the Report Explorer, expand the DataSources node, right-click the node for the data source and select **Add DataSet**. In the DataSet dialog that appears, under **Query** section, go to the **Query String** field and click the drop-down arrow to display the custom query editor. In the custom query editor, click the **Select CSV File** button and select the **Categories.csv** file kept within the project.

Go to the Preview tab of the Designer to view the report with the data pulled from the custom data provider.

The sample consists of following three projects:

CustomDataProvider: It contains following items:

- CsvColumn: This class represents information about fields in the data source.
- CsvCommand: This class provides the IDbCommand implementation for the .NET Framework CSV Data Provider.
- CsvConnection: This class provides an implementation of IDbConnection for the .NET Framework CSV Data Provider.
- CsvDataProviderFactory: This class implements the DataProviderFactory for .NET Framework CSV Data Provider.
- CsvDataReader: This class provides an implementation of IDataReader for the .NET Framework CSV Data Provider.

CustomDataProviderUI: It contains following items:

- CSVFileSelector: This is the form that contains the **Select CSV File** button. This button is displayed in the CSV data provider query editor when you open the **DataSet** dialog and under **Query**, in the **Query String** field and click the drop-down arrow to display the custom query editor. Clicking the **Select CSV File** button allows you to select the **Categories.csv** file that is used as a custom data provider for the sample report.
- QueryEditor: This is the class that reads the content of the specified file and builds the CSV data provider query string.

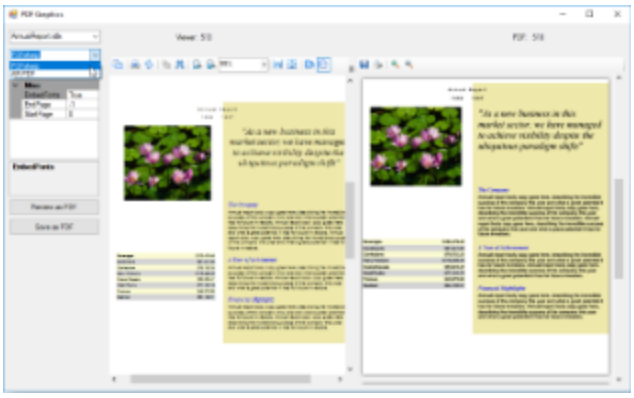
CustomDataProviderUITest: It contains following items:

- Categories.csv: This is the comma separated values (CSV) file that serves as a custom data provider for the sample report. This file is selected in the **Please, select CSV File** dialog that appears when you click the **Select CSV File** button in the **Query String** field under **Query** in the **DataSet** dialog.
- DemoReport.rdlx: The DemoReport.rdlx displays the custom data. This report contains one **Table** data region with the **TextBox** controls, which display the name, the role and the city information of an employee.
- DesignerForm: This is the main form of this sample that appears when you run the sample. On this form, you can connect the sample report to a custom data provider by adding a dataset with a comma separated values (CSV) file. Right-click the form and select **View Code** to see the code implementation for the ActiveReports Designer.
- Grapecity.ActiveReports.config: The configuration file that configures the project to use the custom data provider.
- HelperForm: This form appears on top of the main DesignerForm when you run the sample. This form contains the explanatory text with the steps on how to bind the sample report to the comma separated value (CSV) file. You can

close the Help form by clicking the X button in the upper-right corner of the form.

Custom Pdf Export

This sample shows how to implement simple exports to custom formats (which is not available in ActiveReports right now). The sample uses third-part library to show export to PDF.



Sample Location

Visual Basic.NET

```
..\Samples14\Advanced\PageAndRDL\CustomPdfExport\VB.NET
```

C#

```
..\Samples14\Advanced\PageAndRDL\CustomPdfExport\C#
```

Details

When you run this sample, you see a testing application form, which splits to Windows Forms Viewer and PDF Viewer. You can choose a report and a Pdf export option, click 'Preview as PDF' and view the report in the viewers, and 'Save as PDF' to save the report as Pdf. The sample consists of following projects:

- PdfRendering: It implements **IRenderingExtension** and **IDrawingCanvas** interfaces for customized export to PDF format.
- TestApplication: This is the default start up project to compare and debug custom export.

Custom Resource Locator

The Custom Resource Locator sample demonstrates a custom implementation of the resource locator to load pictures from the user's **Pictures** or **My Pictures** directory. In general, you can use a resource locator in a report to find any resources that a report may require.

Sample Location

Visual Basic.NET

```
..\Samples14\Advanced\PageAndRDL\CustomResourceLocator\VB.NET
```

C#

```
..\Samples14\Advanced\PageAndRDL\CustomResourceLocator\C#
```

Details

When you run this sample, you see the **MainForm** with the list of images from the **Pictures** or **My Pictures** directory. Select any image and click the **Show Report** button. A report with the selected image opens in the **PreviewForm**.

> ⚠ **Caution**: To run this sample properly, you must have image files in your pictures directory. If the directory does not contain any pictures, you should add them to the folder manually.

The sample consists of:

- Resources folder: This folder contains the **Description.rtf** file that contains a summarized content of the resource locator that gets displayed inside the RichTextBox control on the **MainForm** at run time. This folder also contains the **NoImage.bmp** image file that is used if there is no image in the pictures directory.

- DemoReport.rdlx: The DemoReport.rdlx displays the selected image. This report contains two TextBox controls and one Image control, which display the image name, the image type and the image at run time after you click the **Show Report** button on the **MainForm**.

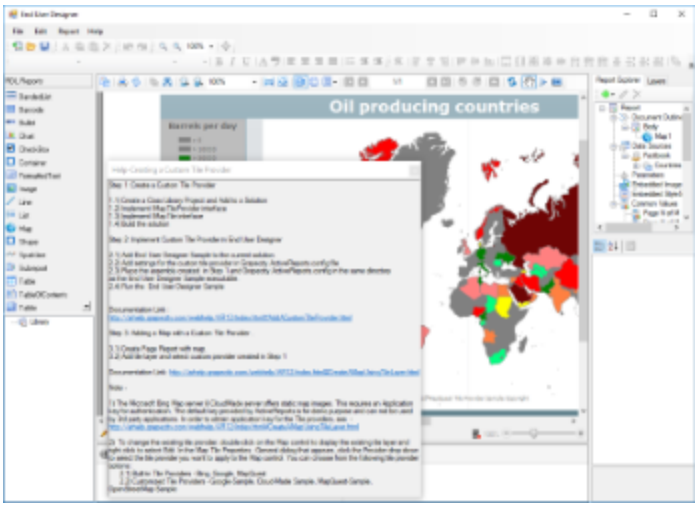
- MainForm: This is the main form of this sample that appears when you run the sample. This form contains the RichTextBox, the ListView and the Button controls. The RichTextBox control displays the summarized information saved in the **Description.rtf** file about the resource locator and the sample. The ListView control gets populated with the images located in the **My Pictures** directory; the Button control is used to generate the report with the selected image. Right-click the form and select **View Code** to see how to load text in the RichTextBox control and images in the ListView control. It also contains code that displays the **DemoReport.rdlx** on the **showReport_Click** event.

- MyPicturesLocator: This file is an internal class that contains code that looks for resources in the **My Pictures** directory.
- PreviewForm: This form uses the ActiveReports **Viewer** control to display the **DemoReport.rdlx** with the selected image. Right-click the form and select **View Code** to see how to load the report into the Viewer.

Custom Tile Provider

The CustomTileProvider sample demonstrates how to create a custom tile provider using **IMapTileProvider** and **IMapTile** interface and configure it in a Map Control which is placed on a RDL report. This sample makes use of two projects - CustomTileProviders and TileProviderEndUserDesigner in a single Visual Studio solution. The CustomTileProviders project contains the tile server configurations for the tile providers, whereas the TileProviderEndUserDesigner project references the created CustomTileProviders project assemblies.

> **Note:** CustomTileProvider is for use with the Professional Edition license only. An evaluation message is rendered when used with the Standard Edition license.



Sample Location

Visual Basic.NET

```
..\Samples14\Advanced\PageAndRDL\CustomTileProvider\VB.NET
```

C#

```
..\Samples14\Advanced\PageAndRDL\CustomTileProvider\C#
```

Details

When you run this sample, the ActiveReports End User Designer appears with an overlaying **Help-Creating a Custom Tile Provider** dialog. This dialog gives you step-by-step instructions to create a new tile provider for a Map control with custom settings.
The End User Designer displays a RDL report containing a Map control with MapQuest set as the default tile provider. To change the existing tile provider, double-click on the Map control to display the existing tile layer and right click to select **Edit**. In the **Map Tile Properties - General** dialog that appears, click the **Provider** drop-down to select the tile provider you want to apply to the Map control. Go to the Preview tab to view the data in the selected tile provider. You can choose from the following tile provider options:

- Google-Sample
- Cloud-Made Sample
- MapQuest-Sample
- OpenStreetMap-Sample

> **Note**: The Microsoft **Bing Map** server offers static map images. This requires an Application key for authentication. The default key provided by ActiveReports is for demo purpose and can not be used by 3rd party applications. In order to obtain a Bing Map Key, ssee HowTo - Create a Bing Map Account and HowTo - Get a Bing Map Key.

The sample consists of two projects:

CustomTileProviders: It contains following classes:

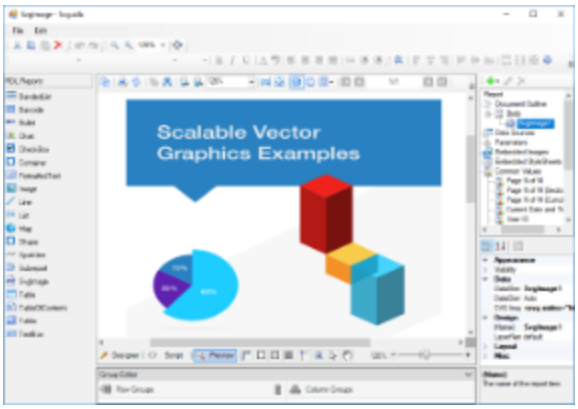
- CloudMadeTileProvider: This class implements the IMapTileProvider interface and contains the settings for the map tile images provided from https://cloudmade.com.
- GoogleMapsTileProvider: This class implements the IMapTileProvider interface and contains the settings for the map tile images provided from https://www.google.com/maps.
- MapQuestTileProvider: This class implements the IMapTileProvider interface and contains the settings for the map tile images provided from https://www.mapquest.com/.
- MapTile: This class represents a single map tile, implementing the IMapTile interface. For more information on ImapTile interface, see Add a Custom Tile Provider.
- OpenStreetMapTileProvider: This class implements the IMapTileProvider interface and contains the settings for the map tile images provided from https://www.openstreetmap.org.
- WebRequestHelper: This class picks the raw data from the tile providers and loads them into the System.IO.MemoryStream class.

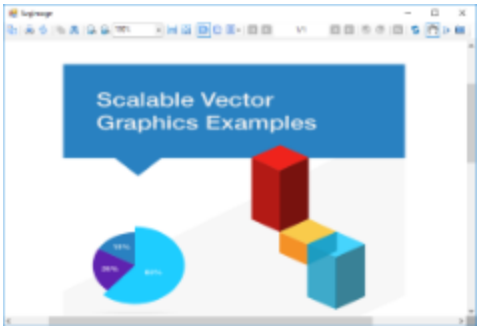
TileProviderEndUserDesigner: It contains following files:

- CustomTileProvider.rdlx: This report contains the Map control that visualizes the oil production in different parts of the world on a virtual earth background. The map control uses the color rule set on a polygon layer to differentiate parts of world as per their oil production capacity. These colors are defined using a color rule which is described in the legend at run time. The report gets the data from **Factbook.rdsx** shared data source.
- DesignerForm.cs: This is the main form that gets displayed when you run the sample. This form uses multiple controls like the ToolStripPanel, ToolStripContainerPanel, SplitContainer, Designer, Toolbox, ReportExplorer and PropertyGrid controls to create a customized End User Designer. It also contains code to load CustomTileProvider.rdlx report into the Designer.
- GrapeCity.ActiveReports.config: This configuration file contains the settings for the various tile providers, and is located in the same folder as the EndUserDesigner.exe file for the tile provider settings to work.
- HelperForm.cs: This form uses the HelperForm class to display a screen containing step-by-step instructions for the user to create a custom tile provider.

Svg Image

This sample illustrates using SVG (Scalable Vector Graphics) as image format in ActiveReports through third-party assembly, **Svg** (SVG Rendering Library, see in NuGet packages). The use of SVG for rendering improves the precision of graphics significantly, so that even complex graphics look sharp and crisp.





Sample Location

Visual Basic.NET

`..\Samples14\Advanced\PageAndRDL\SvgImage\VB.NET`

C#

`..\Samples14\Advanced\PageAndRDL\SvgImage\C#`

Details

When you run this sample, an RDL report with an SVG image is displayed. You may need to install Svg library; the steps are as follows:

1. Open "Package Manager Console".
2. Use "Install-Package -Id Svg" command.
3. Select the project in the Solution Explorer and click Refresh in toolbar.

The sample consists of the following projects:

- SvgImage: It implements **IReportItem** and **IGraphicsRenderer** interfaces to create custom SVG image report item, **SvgImage**.

- SvgDesigner: The designer is inherited from **CustomReportItemDesigner** class. For design-time rendering, **SvgControlGlyph** class implements overriding ControlGlyph property of the designer. In this glyph, the MovableBehavior method implements moving and resizing of the Svg Image control.

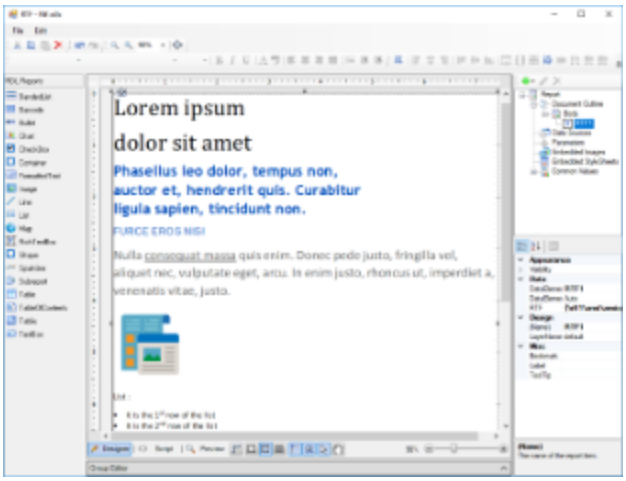
- TestDesignerPro: This is the default startup project. On running this project, an RDL report with an SVG image is displayed on the designer. You can change its properties from the Properties pane. You can also drag and drop and use the SVGImage control available on the toolbox.

- TestViewer: On running this project, an SVG image is rendered in the Windows Forms Viewer.

RTF Control

This sample shows using a custom control, **RichTextBox**, to preview RTF directly in Page/RDL reports without need for conversion to HTML.

Just drag and drop the RichTextBox control to the design area, press **F2** or **Enter** or double click the control to enable edit mode, and input the RTF.



Sample Location

Visual Basic.NET

`..\Samples14\Advanced\PageAndRDL\RtfControl\VB.NET`

C#

`..\Samples14\Advanced\PageAndRDL\RtfControl\C#`

Details

To run the sample, set **TestViewer** or **TestDesignerPro** as startup project.

- TestDesignerPro: This is the default startup project. On running this project, an RDL report with RichTextBox is

displayed on the designer. You can change its properties from the Properties pane or use the control available on the toolbox.
- TestViewer: On running this project, an RichTextBox is rendered in the Windows Forms Viewer.

Note that this sample is implemented based on WinForms RichTextBox control that provides display features similar to WordPad RTF, such as:

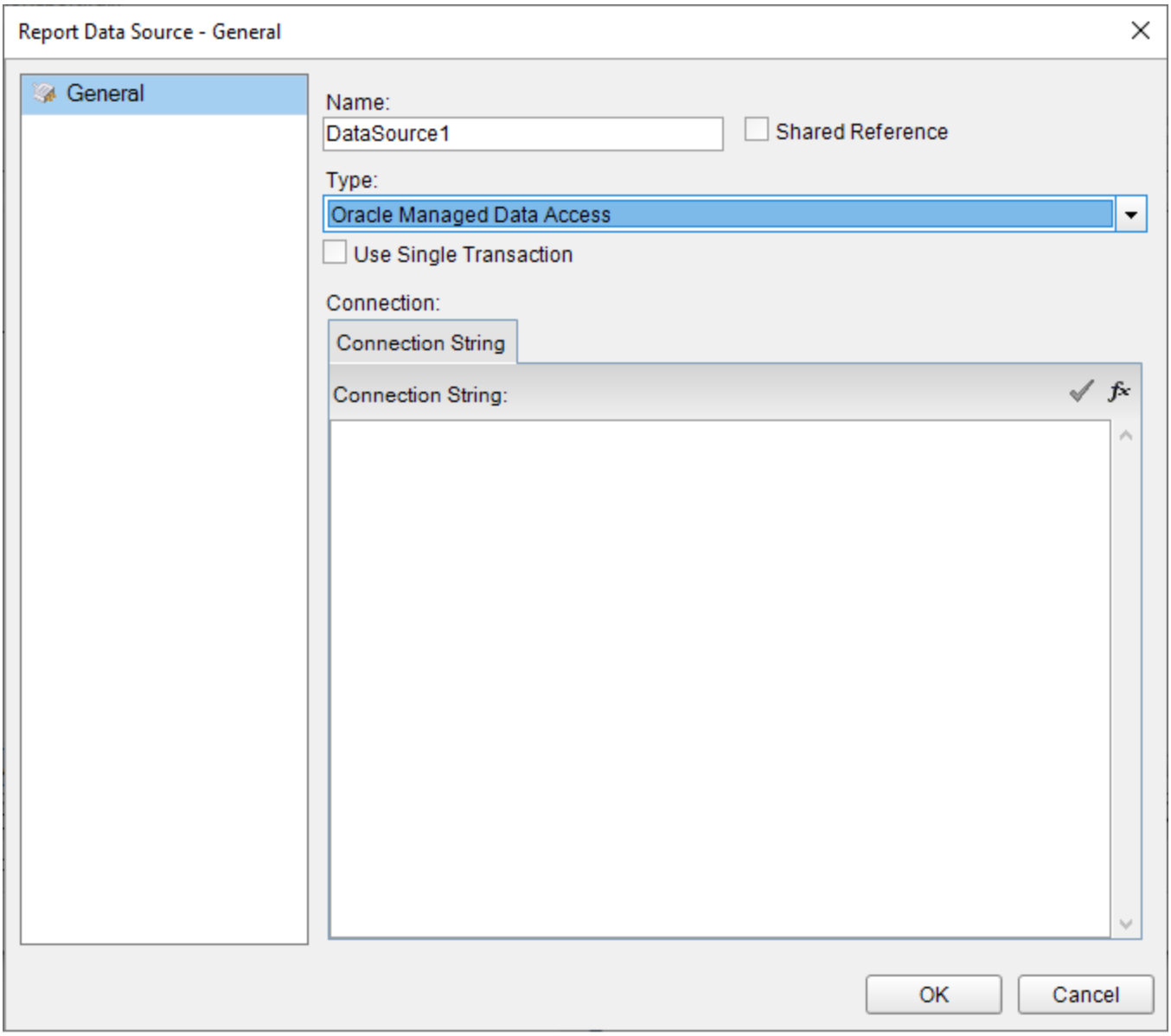
- Images
- Color text and background
- Font size and styles
- Text alignment
- Standard commands like copy/paste and undo/redo

The following hotkeys and shortcuts are supported in edit mode:

- Ctrl + A: Select all content
- Ctrl + C: Copy selected content
- Ctrl + X: Cut selected content
- Ctrl + V: Paste content from clipboard
- Ctrl + B: Toggle bold style of selected text
- Ctrl + I: Toggle italic style of selected text
- Ctrl + U: Toggle underline style of selected text
- Ctrl + T: Toggle strikeout style of selected text
- Ctrl + Z: Undo
- Ctrl + Shift + Z or Ctrl + Y: Redo
- Ctrl + Backspace: Delete the word left to the cursor
- Ctrl + Delete: Delete the word right to the cursor
- Escape: Exit edit mode without saving changes
- Alt + Enter: Exit edit mode with saving changes

Oracle Data Provider

Use this sample if you want to connect to the Oracle Data Provider, which is otherwise not available since System.Data.OracleClient is deprecated.



Sample Location

Visual Basic.NET

`..\Samples14\Advanced\PageAndRDL\OracleDataProvider\VB.NET`

C#

`..\Samples14\Advanced\PageAndRDL\OracleDataProvider\C#`

Details

When you run this sample, a blank DesignerForm for RDL report is displayed. Connect to the Oracle data provider as follows:

1. Add a data source.

2. In the Report Datasource dialog, select **Type** as **Oracle Managed Data Access**.
3. Enter the connection string.

   Sample Oracle Connection String

   provider=ORACLE;data source=in-data-sql/orcl.grapecity.net;user id=user1;password=password@123;

Now, proceed the report designing by pulling the data from Oracle data provider.

The sample consists of following:

TestDesignerPro.csproj: This is the default start up project.

GrapeCity.ActiveReports.config: Located inside the startup project, it is a configuration file that contains the settings for using the oracle data provider:

- **DisplayName** to reference data provider and **Type** to use the oracle custom data provider as mandatory fields
- **AdapterType** implemented in 'OracleConnectionAdapter' class and **SchemaProviderType** implemented in 'GeneralOracleSchemaProvider' class for additional features.

OracleConnectionAdapter.cs: This class provides features related to parameters such as handling multi-value paramaters and parameterized queries.

GeneralOracleSchemaProvider.cs: This class generates DataSchema to enable visual query designer support.

Section Reports

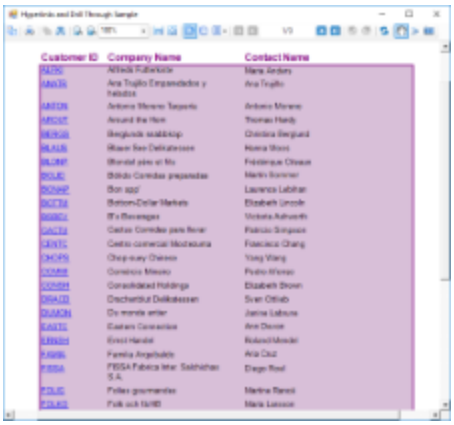
This section discusses following samples describing various features in Section reports:

CustomDrillThrough
: Demonstrates using hyperlinks and the viewer hyperlink event to simulate drill-down from one report to another.

CustomWordExport
: Demonstrates exporting Section report to Word format using third-party assemblies.

Custom Drill Through

The Hyperlinks and DrillThrough sample consists of three reports and a ViewerForm. The reports use the Hyperlink event of the Viewer control to pass a value from the Hyperlink property of a TextBox control to a Parameter value in a more detailed report.



Sample Location

Visual Basic.NET

```
..\Samples14\Advanced\Section\CustomDrillThrough\VB.NET
```

C#

```
..\Samples14\Advanced\Section\CustomDrillThrough\C#
```

Details

When you run this sample, a report displaying bound fields with a link created on CustomerID is displayed in a Viewer control. DrillThrough feature allows users to navigate to another report containing detailed data. Clicking the CustomerID hyperlink takes you to the second report which displays detailed information of the selected CustomerID. On further clicking the OrderID hyperlink the third report displaying the details of the selected order is opened in the Viewer. This feature enables the users to systematically go through the detailed data of the desired CustomerID.

> **Note:** To run this sample, you must have Nwind.mdb downloaded from GitHub in *..\Samples14\Data\NWIND.mdb*.

The sample consists of:

ViewerForm: This form contains only the Viewer control. Right-click the form and select **View Code** to see the code that allows multiple ViewerForms to display for the reports, and see the **Form Load** event for the code that loads the main report into the viewer. See the **Viewer Hyperlink** event for the code that collects a string value from the **Hyperlink** property of the clicked TextBox on the main report and passes it into the **customerID** Parameter of the report **DrillThrough1**, or collects a numeric value and passes it to the orderID Parameter of the report **DrillThrough2**. This code then runs the report with the parameter value and displays it in another instance of the ViewerForm.

DrillThroughMainReport: The main report that is loaded in the ViewerForm by default uses the PageHeader and Detail sections.

- PageHeader section: This section contains three Label controls to serve as column headers for the details, and a CrossSectionBox control. For more information, see Cross Section Controls.

- Detail section: The Detail section has the **BackColor** property set to **Thistle**, and its **RepeatToFill** property set to **True**. This ensures that the background color reaches all the way to the bottom of the page when there is not enough data to fill it. Right click on the form, select **View code** to see the Connection String and SQL Query that provide data for the bound fields. The Detail section has three bound **TextBox** controls that display a list of customer information. Select **CustomerID** and you will see that the **HyperLink** property is not set in the Properties window. To see the code that assigns the data from the TextBox to its HyperLink property, right-click the report and select **View Code**. The **HyperLink** property is set in the **Detail BeforePrint** event.

- PageFooter Section: This section is not in use, so it is hidden by setting the **Visible** property to **False**. This section cannot be deleted, because its related PageHeader section is in use.

  > **Note:** This hyperlink does not work in Preview mode, because it relies on code in the ViewerForm to pass the value to DrillThrough1 report's parameter.

DrillThrough1 Report: This report looks similar to the DrillThroughMain report, but the main difference is that it has a CustomerID parameter in its SQL Query.

- GroupHeader section: Since this report only displays order information for the CustomerID from the clicked

hyperlink, the PageHeader section could have been used, but this report uses the GroupHeader section. To make this section print at the top of each page, the **RepeatStyle** property is set to **OnPage**. This section consists of five label controls to serve as column headers for the Detail section and a CrossSectionBox control.

- Detail section: Right click on the form, select **View code** to see the parameter in the SQL Query that collects the value from the ViewerForm. Parameters in SQL Queries are denoted by the **<%** and **%>** symbols that trigger ActiveReports to add them to the report's Parameters collection. For more information, see Parameters. The Detail section has five bound TextBox controls that display a list of order information for the customer. Select **OrderID** and you will see that the **HyperLink** property is not set in the Properties window. To see the code that assigns the data from the TextBox to its HyperLink property, right-click the report and select **View Code**. The **HyperLink** property is set in the **Detail BeforePrint** event.

- GroupFooter section: This section is not in use, so it is hidden by setting the **Visible** property to **False**. This section cannot be deleted, because it is related GroupHeader section is in use.

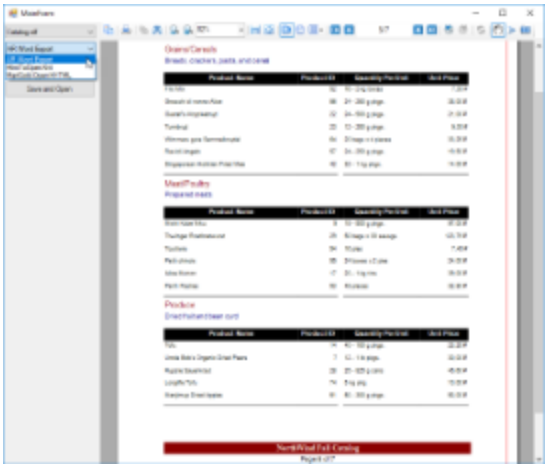
DrillThrough2 Report: Like DrillThrough1, this report has a parameter in a SQL Query, but unlike the other two reports, this one has no hyperlink. It displays order details for the OrderID value passed into it from the clicked hyperlink in DrillThrough1.

- GroupHeader section: Like in the previous report, this section contains Label controls to serve as column headers for the details, and a CrossSectionBox control.

- Detail section: Right click on the form, select **View code** to see the parameter in the SQL Query that collects the value from the ViewerForm.

Custom Word Export

This sample shows how to export Section report to Word (DOCX) format using third party assemblies.



Sample Location

Visual Basic.NET

`..\Samples14\Advanced\Section\CustomWordExport\VB.NET`

C#

```
..\Samples14\Advanced\Section\CustomWordExport\C#
```

Details

When you run this sample, a Windows Form is displayed that prompts you to select the report type and the Word export format type. Please note that we are not going to support any third party assemblies or projects. This sample demonstrates the idea on creating custom exports to satisfy different requirements.

API

The samples in the API folder describe creating reports and implementing various features through code in Page, RDL, and Section reports.

- Page and RDL Reports
- Section Reports

Page and RDL Reports

This section contains:

CreateReport
: This sample demonstrates how to create a page report layout in code. It further shows creating a table control, adding table rows and table columns inside it, adding cells inside the table rows and columns and adding text boxes inside the cells.

DigitalSignaturePro
: This sample demonstrates how to add digital signatures when exporting to PDF format.

Export
: This sample demonstrates how to export Page and RDL reports to different export formats.

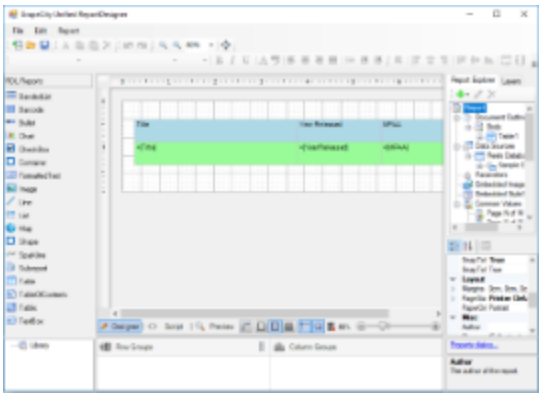
Layer
: This sample demonstrates how to use Layers in a report.

ReportWizard
: This sample demonstrates how to create a custom Report Wizard that allows you to select a report from the list of multiple reports and then allows you to select the data that you want to display in the selected report.

Stylesheets
: This sample demonstrates how to work with embedded and external style sheets in Page and RDL reports.

Create Report

The Create Report sample demonstrates how to create a RDL report using code and display it in the ActiveReports Designer.



Sample Location

Visual Basic.NET

```
..\Samples14\API\PageAndRDL\CreateReport\VB.NET
```

C#

```
..\Samples14\API\PageAndRDL\CreateReport\C#
```

Details

When you run this sample, the ActiveReports Designer appears with a RDL report that is bound to a database. To view the report, go to the Preview tab of the Designer.

> **Note:** To run this sample, you must have access to the **Reels.mdb**. The Reels.mdb file can be downloaded from GitHub: ..\Samples14\Data\Reels.mdb

The sample consists of:

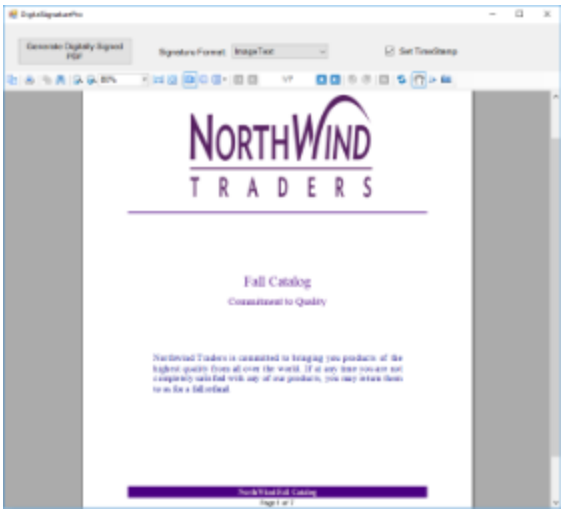
ReportsForm: This is the main form of the sample that contains the Designer, ReportExplorer, PropertyGrid, Toolbox and ToolStrip controls, used to create the ActiveReports Designer at run time. See Creating a Basic End User Report Designer (Pro Edition) for information on creating a basic ActiveReports Designer. Right-click the form and select View Code to see how to set up the Designer and attach the ReportExplorer, PropertyGrid and Toolbox controls to it. It also contains code that loads a layout created in the LayoutBuilder class to a page report object; then loads the page report object to a stream, which is loaded to the Designer.

Constants: This file is an internal class that contains string values that are required for creating a dataset of the report.

Layout: This file is an internal class that contains code for creating a RDL report layout and adding a data source and a dataset to it.

Digital Signature Pro

This sample demonstrates how you can digitally sign or set time stamp for a page/rdl report when exporting it to PDF format using the PDF Rendering Extension.



Sample Location

Visual Basic.NET

```
..\Samples14\API\PageAndRDL\DigitalSignaturePro\VB.NET
```

C#

```
..\Samples14\API\PageAndRDL\DigitalSignaturePro\C#
```

Details

When you run this sample, the Invoice report is displayed in the Viewer control.

Clicking the **Generate Digitally Signed PDF** button in the Viewer toolbar creates a PDF file with a time stamp or digital signatures, based on the settings you have specified in the Viewer toolbar. You can change the content of signatures in the **Signature Format** box and you can add the time stamp to the generated pdf file by checking the **Set TimeStamp** checkbox, in the Viewer toolbar.

When you click the **Generate Digitally Signed PDF** button, a dialog for saving the destination file appears. After you indicate the location for a new PDF file, the PDF report file is created. Digital signature certificates dynamically reference and use GrapeCity.pfx, included in the project. Also, digital signatures dynamically load and use the gc.bmp file that you can find in the Image folder of this sample project.

> **Note:** To run this sample, you must have Nwind.mdb downloaded from GitHub in *..\Samples14\Data\NWIND.mdb*.

- Image folder: This folder stores the gc.bmp file with the GrapeCity logo that digital signatures dynamically load and use.
- Catalog report: It shows a multi page layout spread over four pages in the report. The layout in Page1 and Page2 contains Image, Label and Textbox controls to display introductory text. The layout on Page3 contains a List data region with TextBox controls and a Table to display product details for each product category. The layout on Page4 uses Textbox, Shape and Line controls amongst others to create an Order Form, which a user is to fill manually.
- GrapeCity.pfx: In order to create a digital signature, you must have a valid PKCS#12 certificate (*.pfx) file. For information on creating a self-signed certificate, see the Adobe Acrobat Help topic "Create a self-signed digital ID." You can also create a PFX file from the Visual Studio command line. For more information and links to SDK downloads, see https://www.source-code.biz/snippets/vbasic/3.htm.
- PDFDigitalSignature form: This is the main form of the Sample that uses the ActiveReports **Viewer** control in the bottom section of the form, and a panel docked to the top contains the **Create Digitally Signed PDF** button, the

Signature Format box with the drop-down list and the **Set TimeStamp** checkbox.

Clicking the **Generate Digitally Signed PDF** button opens a dialog for saving the destination file. After you indicate the location for a new PDF file, the PDF report file is created.

The drop-down list of the **Signature Format** box contains the following options.

- Invisible - the invisible pdf digital signature.
- Text - the pdf digital signature that contains text only.
- Image - the pdf digital signature that contains graphics only.
- ImageText - the pdf digital signature that contains text and graphics.

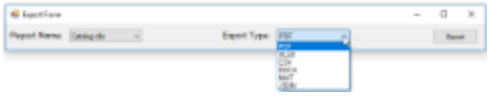
Checking the **Set TimeStamp** checkbox allows you to add the time stamp to the signature of the generated pdf file. The time stamp contains the Time Stamp Server address, its login and password information.

Right-click **PDFDigitalSignature** in the Solution Explorer and select **View Code** to see the code implementation for the pdf digital signature options.

- Resource.resx: This file contains the string for the message box that appears after the PDF file is generated and the string for the message box that appears when the free service limitation is exceeded.

Export

This sample illustrates how to export to different export formats using code. The available export formats are PDF, XLSX, CSV, DOCX, MHT and JSON.



Sample Location

Visual Basic.NET

```
..\Samples14\API\PageAndRDL\Export\VB.NET
```

C#

```
..\Samples14\API\PageAndRDL\Export\C#
```

Details

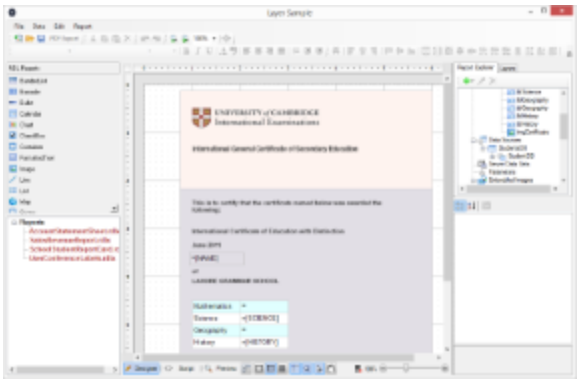
When you run this sample, the Export Form is displayed. The Export Form contains the **Report Name** and **Export Type** combo boxes along with the **Export** button.

The sample consists of **ExportForm**. It exports the selected report in the selected export format.
On running the application, select a report for export in the **Report Name** combo box. In the **Export Type** combo box, you can select one of the following formats for export - PDF, XLSX, CSV, DOCX, MHT and JSON. Clicking the **Export** button opens the **Save As** dialog where you can specify the name of the exported file. By default, the exported file is saved in the **Documents** folder.

Layers

The Layers sample demonstrates how to work with Layers in four different reporting scenarios.

- **Sample Location**
- **Details**
- **ReportForm**
- **Reports Folder**



Sample Location

Visual Basic.NET

```
..\Samples14\API\PageAndRDL\Layers\VB.NET
```

C#

```
..\Samples14\API\PageAndRDL\Layers\C#
```

Details

When you run this sample, the End User Designer shows a list of .rdlx reports on the bottom left of the form. Expand the **Reports** node to view reports under it and double-click a report to load it into the designer. Once the report is loaded in the designer, a **Layers List** window showing all the Layers used in the report is displayed as a tab next to the Report Explorer tab. See Working with Layers for further details on the Layers List window.

ReportForm

This is the main form that appears when you run the Layers sample. This form uses controls such as Designer, ReportExplorer, LayerList, Toolbox, PropertyGrid, ToolStripPanel, ToolStripContentPanel and TreeView to create a customized report designer.

Right-click the form and select **View Code** to see how to set up the report designer. It also contains code that adds reports to the TreeView control, loads a report into the Designer when it is double-clicked in the Reports node, and shows the Layer List window if the report type is Page or RDL.

Reports Folder

AccountStatementSheet.rdlx: This report uses the XML data source connection to provide custom data. Controls such as Image, FormattedTextBox, Table and Label are used to display the statement of an account holder. The FormattedTextBox control contains text in HTML tags to display the content.

This report is contains three Layers besides the Default Layer with the following controls and TargetDevice settings.

Layer Name	Target Device	Description
Default	All	This is a default layer. It does not contain any control.
Layer1	Screen	Contains the Header and Footer section.
Layer2	Screen, Paper	Contains a FormattedTextBox control to display the account information of the account holder.
Layer3	All	Contains a Table data region and an OverflowPlaceHolder control to display the account transactions.

SalesRevenueReport.rdlx: This report uses the Reels shared data source connection to provide the data. The layout of the report uses a Chart data region to graphically display sales and profit for each month and two Table data regions to display the chart data. The first Table data region (Layer2) contains the sales revenue for each month and the second Table data region (Layer3) lists down the detailed figures for the monthly sales along with a DataBar to visually depict the profits.

This report contains three Layers besides the Default Layer with the following controls and TargetDevice settings.

Layer Name	Target Device	Description
Default	All	This is a default layer. It does not contain any control.
Layer1	All	Contains a Chart data region to display the annual sales by month.
Layer2	Paper	Contains a Table data region to display the sales by month. Visibility of this Layer has been set to hidden.
Layer3	Screen, Export	Contains a Table data region to display the detailed sales and a DataBar to depict the profits.

SchoolStudentReportCard.rdlx: This report uses the XML data source connection to provide custom data. This report includes an Image control that acts as a background layout of the report. It also includes a few TextBox controls to display the name and the grades obtained by the student.

This report contains two Layers besides the Default Layer with the following controls and TargetDevice settings.

Layer Name	Target Device	Description
Default	All	This is a default layer. It does not contain any control.
Layer1	Screen, Export	Contains an Image control with an image of a school graduation certificate.
Layer2	All	Contains few TextBox controls to display the name and grades obtained by the student.

UserConferenceLabels.rdlx: This report uses the Nwind shared data source connection to provide data. This report uses the Image, TextBox, BarCode and Label controls to create a layout for the User Conference 2014 ticket.

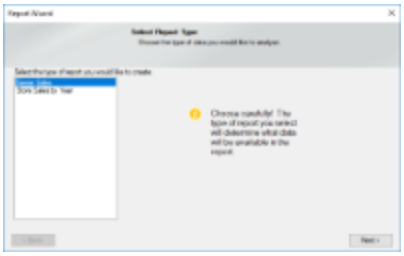
This report contains two Layers besides the Default Layer with the following controls and TargetDevice settings.

Layer Name	Target Device	Description
default	All	This is a default layer. It does not contain any control.

Layer1	Screen, Export	Contains Image controls to display the conference logo and sponsors and Label controls to display conference name and details.
Layer2	All	Contains bound TextBox fields to display the conference attendee details and a BarCode control to display a unique BarCode for each attendee.

Report Wizard

The Report Wizard sample demonstrates how to create and customize a report, using the report wizard.



Sample Location

Visual Basic.NET

```
..\Samples14\API\PageAndRDL\ReportWizard\VB.NET
```

C#

```
..\Samples14\API\PageAndRDL\API\ReportWizard\C#
```

Details

When you run this sample, the form with the Report Wizard appears. On the **Select Report Type** page, select the report type you want to analyze and click the **Next** button.

On the next **Choose grouping options** page of the wizard that appears, you are asked to choose a field for grouping the report data and click the **Next** button. You can also enable the checkbox at the bottom of the page if you like to include the last detail of the report as separate group. If you leave it unchecked, the checkbox automatically adds the last detail to a previous group.

On the next **Select output fields** page of the wizard that appears, you are asked to choose the fields you want to display in your report and click the **Next** button. On the next **Summarization and Review** page, you can review the settings you have selected previously. You can also set the summary options if you want to display a grand total or a sub-total in the report. Finally, when you click the **Finish** button, the GrapeCity **Unified ReportDesigner** appears and displays the report.

The sample consists of following items:

MetaData folder: This folder contains two internal classes, **FieldMetaData** and **ReportMetaData**. The FieldMetaData class contains information on the fields used in the report string values. This file provides this information when required by the application. Similarly, the ReportMetaData class contains information on the reports used in this sample. This file provides this information when required by the application.

Resources folder: This folder contains images and icons used by the Report Wizard API.

UI folder:

- WizardSteps: This folder contains templates of the Report Wizard pages, which get displayed inside the Panel

control on the WizardForm at run time.

- DesignerForm: This form appears when you click the **Finish** button on the last page of the Report Wizard. This form uses the ToolStripPanel, ToolStripContentPanel, Designer, Toolbox, ReportExplorer and a PropertyGrid controls to create the Unified ReportDesigner.
- Right-click the form and select **View Code** to see how to set the designer and create a blank page report. It also contains code that attaches the Toolbox, ReportExplorer and PropertyGrid controls to the Designer, inserts DropDown items to the ToolStripDropDownItem, sets their functions, and checks for any modifications that have been made to the report in the designer.
- DragDropListBox: This file contains code to override methods that enable and handle the drag-and-drop feature in the ListBox, which appears on the wizard page where you select the grouping and the output fields. Thus, instead of selecting a field and clicking the **Add** button in the ListBox control, you can directly drag fields as well.
- TipControl: This file contains the design of the Tip that appears on the first page of the report wizard.
- WizardDialog: This is the main form that appears when you run the sample. It uses the PictureBox, two Labels, Panel and two Button controls to create the Report Wizard. The PictureBox displays the logo while the two Label controls display the Page Title and Page Description respectively. The Panel control loads the design of different pages of the Report Wizard. The two Button controls handle the last page and next page functions. Right-click the form and select **View Code** to see how to define functions of different controls used on the form.

Constants: This is an internal class that contains fields in string values that can be summarized.

GenreSales.rdlx-master: This is the master report for the **Genre Sales** report. It uses the Image, two Textbox and ContentPlaceholder controls to design the report. The Image control displays the logo on the top of the report while the two Textbox controls display the report execution time and page number information respectively. The ContentPlaceholder control displays its content report.

LayoutBuilder: This is the internal class file that contains code for creating the layout of both child reports and loading data in it.

Reports.xml: This XML file is used as a database to provide data for the reports in this sample.

ReportWizardState: This is the internal class file that contains code for handling the UI of the Report Wizard.

StoreSales.rdlx-master: This is the master report for the **Store Sales** report. It uses the Image, two Textbox and ContentPlaceholder controls to design the report. The Image control displays the logo on the top of the report while the two Textbox controls display the report execution time and page number information respectively. The ContentPlaceholder control displays its content report.

Stylesheets

The Stylesheets sample demonstrates how to work with embedded and external style sheets in Page and RDL reports.

Sample Location

Visual Basic.NET

```
..\Samples14\API\PageAndRDL\Stylesheets\VB.NET
```

C#

```
..\Samples14\API\PageAndRDL\Stylesheets\C#
```

Details

When you run this sample, the main **StyleSheetsForm** interface displaying the following options appears:

1. **Choose Report:** Select the type of report you want to display in the Viewer.
2. **Choose Style:** Select a style sheet; embedded or external to apply to the report.

Click on **Run Report** button to load the report with the selected options in the Viewer. See Styles for more information about style sheets.
StyleSheetsForm: This is the main form of the sample that displays reports in the ActiveReports **Viewer** control in the bottom section of the form, and options such as Choose report, Choose style and Run report button at the top of the form.

DeliverySlip.rdlx: This is a Page report, that contains TextBox, Label, Container controls and two Table data regions to display the invoice information. The report uses Theme1.rdlx-theme file and BaseStyle as an embedded style sheet. Some TextBox controls on the report also use the Sum function to display the total price information for each invoice. This report uses the **Seikyu2** shared data source.

ReorderList.rdlx: This is an RDL report, that contains Table data region to display data from **Reels** shared data source. The Reels logo in the report is embedded within the Reels.rdlx-theme.

External Stylesheet: Following external style sheets are provided in this folder:

External Stylesheet Location - ..\Samples14\API\PageAndRDL\Stylesheets\Reports

- BaseStyle.rdlx-styles
- FaxSheetStyle.rdlx-styles
- HighContrastStyle.rdlx-styles
- ModernStyle.rdlx-styles

Section Reports

This section discusses following samples:

Charting
: This sample demonstrates chart types used in different scenarios, in both bound and unbound modes.

Cross Section Controls
: This sample demonstrates the use of the cross section lines and boxes.

Cross Tab Report
: This sample demonstrates using unbound data, conditional highlighting and distributing data across columns to create a cross-tab view and data aggregation.

Custom Annotation
: This sample demonstrates adding the Custom Annotation button to the report Viewer toolbar and adding a new annotation to the report.

Digital Signature Pro
: This sample demonstrates how to sign digitally or set time stamp for a section report when exporting it to PDF format.

Export
: This sample demonstrates how to export to different export formats using code.

Inheritance
: This sample demonstrates using the method that inherits a report at run time and design time.

Print Multiple Pages per Sheet
: This sample demonstrates printing a document with multiple pages per sheet by using the common PrintDocument class of the NET.Framework.

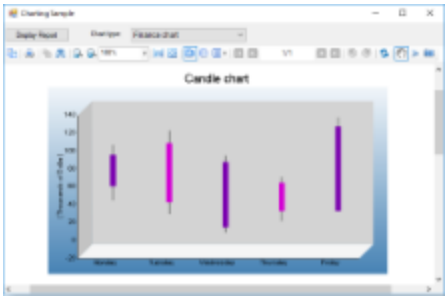
Style Sheets
: This sample demonstrates changing styles at run time to provide a different look to a same report.

Sub Report
: This sample demonstrates using subreports in an ActiveReports report.

Summary
: This sample demonstrates how to display summarized data in a section report.

Charting

The Charting sample provides an option to choose from various chart types and a button to display the selected chart in a Viewer control.



Sample Location

Visual Basic.NET

```
..\Samples14\API\Section\Charting\VB.NET
```

C#

```
..\Samples14\API\Section\Charting\C#
```

Details

Chart type combobox

Select from the following ChartType options.

- 2D Bar Chart - Use this chart to compare values of items across categories.
- 3D Pie Chart - Use this chart to display data in 3D format to depict how percentage of each data item contributes to a total percentage. Selecting this ChartType provides an option to set the Direction of rotation of 3D Pie chart to Clockwise or Counterclockwise.
- 3D Bar Chart - Use this chart to compare values of items across categories and to display the data in a 3D format.
- Finance Chart -Use this chart to display the stock information using High, Low, Open and Close values. The size of the wick line is determined by the High and Low values, while the size of the bar is determined by the Open and Close values.
- Stacked Area Chart - Use this chart to demonstrate how each value contributes to the total.

Report Display button
Click this button to display the selected chart type in a Viewer control.

> **Note:** To run rpt2DBar, rpt3DPie and rpt3DBar report, you must have access to the Nwind database. The NWIND.mdb file can be downloaded from GitHub: ..\Samples14\Data\NWIND.mdb.

ViewerForm
The ViewerForm contains the **Viewer** control, with the **Dock** property set to **Fill**. This enables the viewer to automatically resize along with the form. Right-click the form and select **View Code** to see the code used to run the report and display it in the viewer.

rpt2DBar report
Displays bar chart on a report. Retrieves the data to be displayed in a chart from Orders table in **Nwind.mbd** database. Settings for chart data source can be done using the **Chart DataSource** dialog.

rpt3DBar report
Displays 3D bar chart on a report. Retrieves the data to be displayed in a chart from Orders table in **Nwind.mbd** database. Generates a DataSet for the chart in ReportStart event and sets it in DataSource property of Chart control.

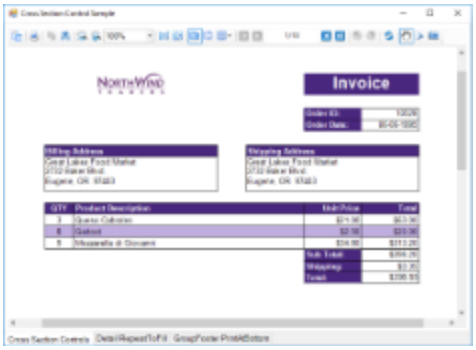
rpt3DPie report
Displays 3D pie chart on a report. Retrieves the data to be displayed in a chart from each of the Employees, Categories, Products, Orders, Order Details tables in **Nwind.mdb** database. Generates a DataTable for the chart in a ReportStart event and sets it in DataSource property of Chart control. Rotational direction of 3D pie chart can be set to **Clockwise** or **Counterclockwise**.

rptCandle report
Displays candle chart on a report. Chart data is set at design time using **DataPoint Collection Editor**. DataSource property is not used for this chart.

rptStackedArea report
Displays stacked area chart on a report. Chart data is set at design time using **DataPoint Collection Editor**. DataSource property is not used for this chart.

Cross Section Controls

This CrossSectionControls sample displays the invoice report of a company in detail. This sample uses the CrossSectionBox and CrossSectionLine controls to demonstrate the lines and display the Invoice report in a tabular form. It includes a **ViewerForm** with three tabs, three **Viewer** controls and an **Invoice** report to highlight several report features. Run the project and click the tab to see these features in action.



Sample

Visual Basic.NET

```
..\Samples14\API\Section\CrossSectionControls\VB.NET
```

C#

```
..\Samples14\API\Section\CrossSectionControls\C#
```

Details

When you run this sample, a form with three different tab options and each tab option displaying a report in a Viewer control is displayed. Click any tab option at the bottom of the form to display the selected tab feature applied to the report in a Viewer control. Select from the following tab options.

Cross Section Controls
Draws table style gridlines easily through multiple sections without any gap.

Detail RepeatToFill
The **Detail RepeatToFill** tab has the **RepeatToFill** property set to **True**. This ensures that the formatting (alternating purple and white rows and CrossSection controls) fills space as needed to keep the same layout between pages.

GroupFooter PrintAtBottom
The GroupFooter section has the PrintAtBottom property set to **True**. This pulls the GroupFooter section to the bottom of the page, just above the PageFooter section.

> **Note:** To run this sample, you must have access to the **Nwind.mdb**. The NWIND.mdb file can be downloaded from GitHub: ..\Samples14\Data\NWIND.mdb.

- ViewerForm: The ViewerForm has three tabs-**Cross Section Controls, Detail RepeatToFill, GroupFooter PrintAtBottom**, each with an ActiveReports Viewer control on it. Right-click the form and select **View Code** to see the code used to change the Invoice report's section properties at run time.
- Invoice Report: The Invoice report demonstrates the usage of the following features.

- PageHeader Section: This section contains Shape, Label and TextBox controls. The **Shape** control provides a border around the **Order ID** and **Order Date** fields and labels. The orderDateTextBox has the **OutputFormat** property set to **d** to display a short date. The **Label** controls use the **BackColor**, **ForeColor**, and **Font** properties to add a distinctive style to the report.
- customerGroupHeader: The CrossSectionBox control is hosted in the GroupHeader section, and spans the Detail section to end in the GroupFooter section, forming a rectangle around the details of the invoice at run time. Three of the CrossSectionLine controls are hosted in the GroupHeader section, and span the Detail section to end in the GroupFooter section, forming vertical lines between columns of invoice details at run time.

> **Note:** If you try to drop a CrossSectionBox or CrossSectionLine control into a section other than a header or footer, the mouse pointer changes to unavailable, and you cannot drop the control.

Two of the TextBox controls use a **CalculatedField** in the **DataField** property.

> **Tip**: In the Report Explorer, expand the **Fields** node, then **Calculated** to see all of the calculated fields. Select **BillingAddress** or **ShippingAddress** to take a look at the **Formula** used in the Properties window.

The **Line** control is used below the column header labels to draw a horizontal line across the width of the report. (It is not visible at design time unless you make the Height of the GroupHeader section larger.) The **DataField** property of the customerGroupHeader section is set to the **OrderID** field, so that the section (followed by related details and GroupFooter) prints once per order.

- Detail section: This section contains four bound TextBox controls. The four **TextBox** controls display each row of data associated with the current GroupHeader OrderID. The **OutputFormat** property of the UnitPrice and Total fields is set to **C** to display currency. The **Line** control is used below the TextBox controls to draw horizontal lines across the width of the report under each row of data. (It is not visible at design time unless you make the Height of the Detail section larger.) Right-click the report and select **View Code** to see the code used in the **Detail Format** event to create a colored bar (in this case, purple bar) report by alternating the **BackColor** property of the section. Click the Data Source Icon on the **Detail** band to view the **Connection String** used in the report.
- customerGroupFooter section: This section has the **NewPage** property set to **After** so that a new page is printed for each OrderID (the associated GroupHeader's DataField). The Subtotal TextBox uses the following properties.
  - The **DataField** property uses a **Total** CalculatedField.
  - The **SummaryFunc** property is set to **Sum**, to add the values of the field in the detail section.
  - The **SummaryGroup** property is set to the name of the **customerGroupHeader**, to reset the summary value each time the GroupHeader section runs.
  - The **SummaryRunning** property is set to **Group** so that the value accumulates for the group rather than for the entire report or not at all.
  - The **SummaryType** property is set to **GrandTotal**.

  Right-click the report and select **View Code** to see the code used in the **customerGroupFooter Format** event to calculate the value for the Grand Total TextBox, and to format it as currency.

- PageFooter section: The **ReportInfo** control uses a **FormatString** property value of **Page {PageNumber} of {PageCount}**, one of the preset values you can use for quick page numbering.

> **Tip:** In order to easily select a control within the report, in the Report explorer, expand the section node and select the control. The control is highlighted in the Report Explorer and on the report design surface.

Cross Tab Report

The CrossTabReport sample displays hierarchical information in a **cross tabular** structure. This sample consists of a StartForm with an ActiveReports Viewer control and a ProductWeeklySales report.

Sample Location

Visual Basic.NET

```
..\Samples14\API\Section\CrossTabReport\VB.NET
```

C#

```
..\Samples14\API\Section\CrossTabReport\C#
```

Details

When you run the sample, a report displaying a list of weekly sales for current week, month, quarter or year for each product category is displayed in the Viewer control. The values highlighted in red represent negative values.

> **Note:** To run this sample, you must have access to the **Nwind.mdb**. The NWIND.mdb file can be downloaded from GitHub: ..\Samples14\Data\NWIND.mdb.

StartForm

The Viewer control has the **Dock** property set to **Fill**. This ensures that the viewer resizes along with the form at run time. Right-click the form and select **View Code** to see the code used to run the report and display it in the viewer.
ProductWeeklySales Report:

ProductWeeklySales Report

This report features a number of accumulated values using summary function property settings and calculated values in the code.The summary function displays a list of summary values only for weekly sales total in group footer without displaying the Detail section. By setting the **Height** property of Detail section to **0** and/or hiding the Detail section by setting **Visible** property to **False**, you can create a summary report that only displays sections other than the Detail section.
Based on the order date of Detail section data, you can determine whether it will be included in a week, month, quarter, year or previous year's quarter and set the values by sorting in unbound fields. Value of each unbound field is automatically summarized by each Field placed in the group footer section.

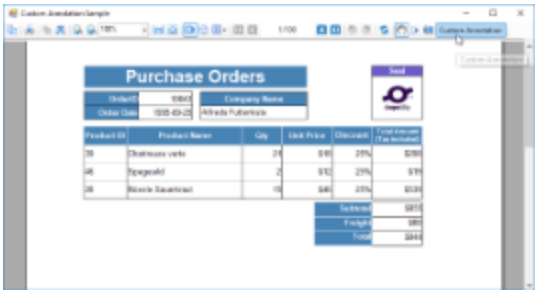
- ReportHeader section: This section of the report features static controls including Labels, a Picture, a Line, and a Shape control. The report header prints only once on the first page of the report, thus report title, company information, and a logo are added in this section.
- PageHeader section: The page header section contains static Label controls that print at the top of each page and serve as column headers for the group header sections.
- ghCategory section: This group header section has the DataField property set to CategoryName. This setting, along with data sorted by the same field, produces a report grouped by category. The section contains one bound TextBox control to display the category name at the beginning of each group. The section's UnderlayNext property is set to True so that the category prints to the left of the top line of data instead of above it.
- ghProduct section: Although this group header contains no controls and is hidden by setting the Height property to 0 and Visible property to False, it still performs two important functions. First, the DataField property is set to ProductName, to sort the data inside each category by product, and second, the related group footer section displays the bulk of the data for the report.
- Detail section: The detail section of this report is hidden by setting the Height property to 0 and Visible property to False, but it does contain four bound fields whose values are used in the code. In the Detail Format event, the value from the hidden txtDetProduct TextBox is collected and passed to the _sProductName variable. For more information on section events, see Section Report Events.
- gfProduct section: This group footer section displays the bulk of the data for the report in TextBox controls that have values passed in code, or are bound to fields from the report's **Fields** collection (see FetchData and DataInitialize events in the code) using the **DataField** property. In the **gfProduct Format** event for the inner group footer section, the product name collected from the Detail Format event is passed to the **txtProduct** TextBox. The **Value** for the **txtPQTDChange** TextBox is calculated by subtracting the prior year's quarter-to-date sales figure from the current quarter-to-date sales figure. The **BackColor** of the **txtPQTDChange** TextBox is set to **Red** if the value is negative. The total units and sales for each product is summarized using the following properties.
    - **SummaryFunc**: Sum (the default value)
      Adds values rather than counting or averaging them.
    - **SummaryGroup**: ghProduct
      Summarizes the values that fall within the current product group.
    - **SummaryRunning**: None (the default value)
      Ensures that this value is reset each time the product group changes.
    - **SummaryType**: SubTotal
      Summarizes the current group rather than a page or report total.
- gfCategory section: This group footer section displays totals of the gfProduct data in TextBox controls that have values passed in code, or are bound to fields from the report's **Fields** collection (see FetchData and DataInitialize events in the code) using the **DataField** property. The total units and sales for each category is summarized using the following properties.
    - **SummaryFunc**: Sum (the default value)
      Adds values rather than counting or averaging them.
    - **SummaryGroup**: ghCategory
      Summarizes the values that fall within the current category group.
    - **SummaryRunning**: None (the default value)
      Ensures that this value is reset each time the category group changes.
    - **SummaryType**: SubTotal
      Summarizes the current group rather than a page or report total.

- PageFooter section: This section is not used, so it is hidden by setting the **Height** property to **0** and/or **Visible** property to **False**. Otherwise, it would print at the bottom of each page. The section cannot be deleted, because the related PageHeader section is in use.

- ReportFooter section: This section is not used, so it is hidden by setting the **Height** property to **0** and/or **Visible** property to **False**. Otherwise, it would print once at the end of the report. The section cannot be deleted, because its related ReportHeader section is in use.

Custom Annotation

The Custom Annotation sample demonstrates how to add the **Custom Annotation** button to the Viewer toolbar and depicts the method to add any annotation (seal image in this case) to the report. Only one annotation can be used per page.



Sample Location

Visual Basic.NET

```
..\Samples14\API\Section\CustomAnnotation\VB.NET
```

C#

```
..\Samples14\API\Section\CustomAnnotation\C#
```

Details

When you run the sample, the report appears in the Viewer control. The Viewer control toolbar contains the **Custom Annotation** button that opens the report annotation.

> **Note:** To run this sample, you must have access to the **Nwind.mdb**. The NWIND.mdb file can be downloaded from GitHub: ..\Samples14\Data\NWIND.mdb.

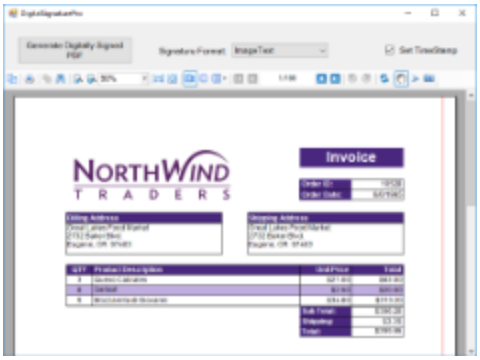
- AnnotationForm: Add **Custom Annotation** button in Form_Load event. The behavior on clicking the Custom Annotation button is mentioned in the description of the Click event.
- Annotation report:
  - **ghOrderID section**
    Product order receipt is grouped according to OrderID.
  - **Detail section**
    Use RepeatToFill property to output empty rows till the end and perform page break group wise.
  - **GFOrderID section**
    This group footer section displays the bulk of the data for the report in TextBox controls that have values passed in code, or are bound to fields from the report's **Fields** collection (see FetchData and DataInitialize events in the code) using the DataField property. The total units and sales for each product are summarized using the following properties:
    - **SummaryFunc:** Sum (the default value) adds values rather than counting or averaging them.

> ⚠ **Caution:** SummaryFunc has no effect unless the SummaryType property is set to either SubTotal or GrandTotal.

  - **SummaryGroup:** ghOrderID summarizes the values that fall within the current order id.
  - **SummaryRunning:** None (the default value) ensures that this value is reset each time the order id changes.
  - **SummaryType:** SubTotal summarizes the current group rather than a page or report total.
- Resources folder: This folder holds the icon used for adding annotation (seal image) to the report.
- Resource1.resx: This file contains the string for the message box that appears when **Custom Annotation** button is clicked again to add the annotation.

Digital Signature Pro

This sample demonstrates how you can digitally sign or set time stamp for a section report when exporting it to PDF format.



Sample Location

Visual Basic.NET

```
..\Samples14\API\Section\DigitalSignaturePro\VB.NET
```

C#

```
..\Samples14\API\Section\DigitalSignaturePro\C#
```

Details

When you run this sample, the Invoice report is displayed in the Viewer control.

Clicking the **Generate Digitally Signed PDF** button in the Viewer toolbar creates a PDF file with a time stamp or digital signatures, based on the settings you have specified in the Viewer toolbar. You can change the content of signatures in the **Signature Format** box and you can add the time stamp to the generated pdf file by checking the **Set TimeStamp** checkbox, in the Viewer toolbar.

When you click the **Generate Digitally Signed PDF** button, a dialog for saving the destination file appears. After you indicate the location for a new PDF file, the PDF report file is created. Digital signature certificates dynamically reference and use GrapeCity.pfx, included in the project. Also, digital signatures dynamically load and use the gc.bmp file that you can find in the Image folder of this sample project.

> **Note:** To run this sample, you must have access to the Nwind database. The NWIND.mdb file can be downloaded from GitHub: ..\Samples14\Data\NWIND.mdb.

- Image folder: This folder stores the gc.bmp file with the GrapeCity logo that digital signatures dynamically load and use.
- Invoice report: The Invoice report is the Sample report that uses three GroupHeader sections, a Detail section and a GroupFooter section as well as a label in the PageFooter section to display data. See The Bound Data Sample topic for details on the Invoice report.
- GrapeCity.pfx: In order to create a digital signature, you must have a valid PKCS#12 certificate (*.pfx) file. For information on creating a self-signed certificate, see the Adobe Acrobat Help topic "Create a self-signed digital ID." You can also create a PFX file from the Visual Studio command line. For more information and links to SDK downloads, see https://www.source-code.biz/snippets/vbasic/3.htm.
- PDFDigitalSignature form: This is the main form of the Sample that uses the ActiveReports **Viewer** control in the bottom section of the form, and a panel docked to the top contains the **Create Digitally Signed PDF** button, the **Signature Format** box with the drop-down list and the **Set TimeStamp** checkbox.

  Clicking the **Generate Digitally Signed PDF** button opens a dialog for saving the destination file. After you indicate the location for a new PDF file, the PDF report file is created.

  The drop-down list of the **Signature Format** box contains the following options.

  - Invisible - the invisible pdf digital signature.
  - Text - the pdf digital signature that contains text only.
  - Image - the pdf digital signature that contains graphics only.
  - ImageText - the pdf digital signature that contains text and graphics.

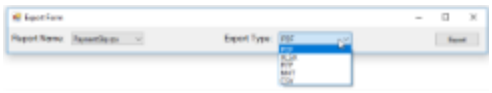
  Checking the **Set TimeStamp** checkbox allows you to add the time stamp to the signature of the generated pdf file. The time stamp contains the Time Stamp Server address, its login and password information.

  Right-click **PDFDigitalSignature** in the Solution Explorer and select **View Code** to see the code implementation for the pdf digital signature options.

- Resource.resx: This file contains the string for the message box that appears after the PDF file is generated and the string for the message box that appears when the free service limitation is exceeded.

Export

This sample illustrates how to export to different export formats using code. The available export formats are PDF, XLSX, RTF, MHT and CSV.



Sample Location

Visual Basic.NET

```
..\Samples14\API\Section\Export\VB.NET
```

C#

```
..\Samples14\API\Section\Export\C#
```

Details

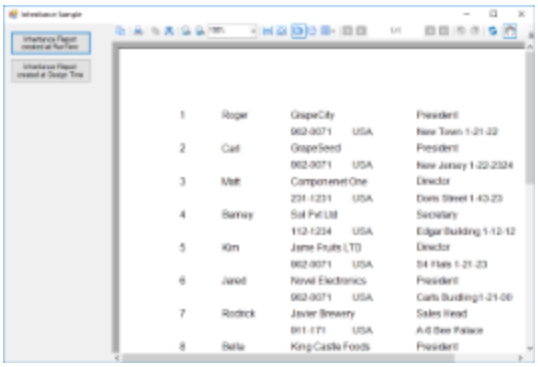
When you run this sample, the Export Form is displayed. The Export Form contains the **Report Name** and **Export Type** combo boxes along with the **Export** button.

The sample consists of **ExportForm**. It exports the selected report in the selected export format.

On running the application, select a report for export in the **Report Name** combo box. In the **Export Type** combo box, you can select one of the following formats for export - PDF, XLSX, RTF, MHT and CSV. Clicking the **Export** button opens the **Save As** dialog where you can specify the name of the exported file. By default, the exported file is saved in the **Documents** folder.

Inheritance

This sample explains the method to inherit a report at run time and design time. The Inheritance sample solution is composed of two classes - the parent class and the child class for both inheritance at run time and design time.



Sample Location

Visual Basic.NET

```
..\Samples14\API\Section\Inheritance\VB.NET
```

C#

```
..\Samples14\API\Section\Inheritance\C#
```

Details

When you run the sample, the report is created and displayed, providing you with the choice of two options - **Inheritance Report created at RunTime** and **Inheritance Report created at Design Time**. By clicking a button on the form, you get a report that inherits another class at run time or at design time.

Inheritance Report created at RunTime
The rptInheritBase class is the inheritance class for the generated report when the **Inheritance Report created at RunTime** button is clicked on the form. The rptInheritBase class inherits the SectionReport class of the GrapeCity.ActiveReports namespace as parent class. It is possible to use DataInitialize event or FetchData event in this class and also possible to load csv file and set values for csv files. It defines the CsvPath property which gets csv file path. The rptInheritChild class inherits the rptInheritBase class and is a class which only defines the report design. By adding the event handler for inheritance in the constructor and setting for csv file in CsvPath property, it is possible to perform rendering of data executed by event of BaseReport which is the inheritance class.

Inheritance Report created at Design Time

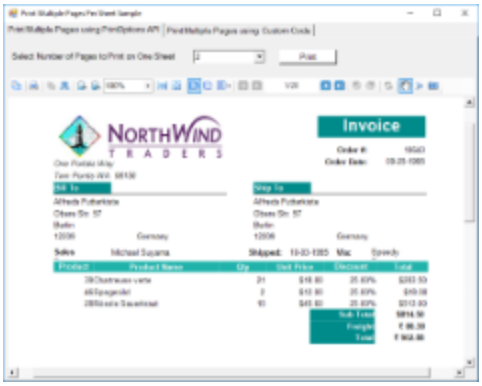
The rptDesignBase class is the inheritance class for the generated report when the **Inheritance Report created at Design Time** button is clicked on the form. The rptDesignBase class inherits SectionReport class of the GrapeCity.ActiveReports namespace as parent class. Using this class, you can place any control (ReportInfo controls etc. to display report title, page number, page count) you wish to inherit in PageHeader section and PageFooter section. The rptDesignChild class is inherited from rptDesignBase class. It only defines the design of Detail section. PageHeader section and PageFooter section use the design of rptDesignBase class which is an inherited class. DataSource setting can be performed from rptDesignChild class.

> ⚠ **Caution:** If you have not run the project even once, an error occurs when you try to open the report designers of the inherited classes rptInheritChild or rptDesignChild from the solution explorer. In case this error occurs, **Build** the project once before opening the report.

- ViewerForm: Creates an instance of specified report and display the report in Viewer control.
- rptDesignBase: The rptDesignBase class that defines the layout of PageHeader section and PageFooter section.
- rptDesignChild: The rptDesignChild designer defines the layout on Detail section and sets the value of DataField property of the controls placed in Detail section. Also set the data source to output using **Report Data Source** dialog.
- rptInheritBase: The class to set values for data field and rendering of csv file using DataInitialize event FetchData event.
- rptInheritChild: The designer that sets layout for each control and it's DataField property.

Print Multiple Pages per Sheet

The PrintMultiplePagesPerSheet sample demonstrates how you can print a document with multiple pages per sheet using the common PrintDocument class or PrintOptions class from .NET Framework. This sample project consists of the PrintMultiplePagesForm and the Invoice report.



Sample Location

Visual Basic.NET

```
..\Samples14\API\Section\PrintMultiplePagesPerSheet\VB.NET
```

C#

```
..\Samples14\API\Section\PrintMultiplePagesPerSheet\C#
```

Details

When you run this sample, you will see the PrintMultiplePagesForm with the Invoice report. On this form, you can select

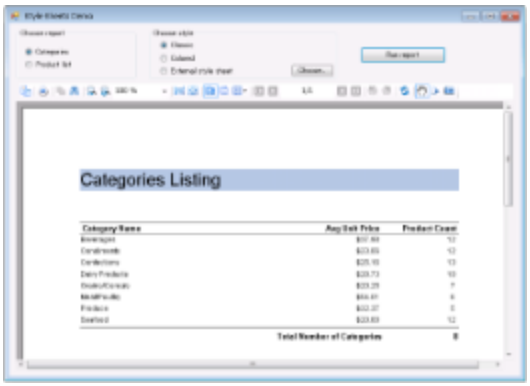
the number of pages to be printed on each sheet using the **Select Numer of Pages to Print on One Sheet** ComboBox. You can also select from **PrintMultiple Pages using PrintOptions API** and **PrintMultiple Pages using Custom Code** tabs options and click the **Print** button on each of these tab to print the selected number of pages in one sheet.

> **Note:** To run this sample, you must have access to the **Nwind.mdb**. The NWIND.mdb file can be downloaded from GitHub: ..\Samples14\Data\NWIND.mdb.

- PrintMultiplePagesForm: This form contains the ActiveReports **Viewer** control. The **Dock** property of the viewer is set to **Fill** so that it resizes automatically with the form at run time. The top section of Viewer contains a panel in which two tabs, ComboBox control, Label and two Print buttons are placed. **ComboBox** control lets you select the number of pages per sheet (2,4 or 8)and the **Print** button in **PrintMultiple Pages using PrintOptions API** and **PrintMultiple Pages using Custom Code** tab, print the selected number of pages in one sheet. The form also has two dialogs - dlgPrint and PrintDocument which assist in displaying the Print dialog box and printing the document.
  Right-click and select **View Code** to see the code that displays the Invoice report when the form loads. Also the code demonstrates the different ways of printing a document - the **Print** button in **PrintMultiple Pages using Custom Code** tab uses the PrintDocument class and the **Print** button in **PrintMultiple Pages using PrintOptions API** tab uses the PrintOptions class.
- Invoice report: The Invoice report uses a PageHeader section, GroupHeader section, Detail section, GroupFooter section as well as a PageFooter section to display data in a Label control.

Style Sheets

This sample demonstrates how you can change styles at run time to provide a different look to a same report. The project includes two reports, three report styles and a form containing the ActiveReports Viewer control and other controls that allow you to select any combination of styles and reports.



Sample Location

Visual Basic.NET

```
..\Samples14\API\Section\Stylesheets\VB.NET
```

C#

```
..\Samples14\API\Section\Stylesheets\C#
```

Details

Choose Report
Choose between the type of report, Categories and Product List, you want to display in the Viewer control.

Choose Style
Choose between **Classic**, **Colored** and **External style sheet** options to apply the style to the selected report.
Clicking the **Choose** button option for External style sheet displays the **Open** dialog that shows only ***.reportstyle** files, and passes the selected reportstyle path and file name string to the externalStyleSheet variable.

Run Report button
Click this button to display the selected report with the applied style in a Viewer control. Clicking this button creates an SectionReport object, assigns the selected report to it, and assigns a path and file name string to the styleSheet variable. It then assigns the style sheet to the report using the LoadStyles(styleSheet) method, runs the report, and displays it in the viewer.

The sample consists of:

- Report Style Sheets: Look in Solution Explorer to see several *.reportstyle files. These are XML-based files that hold styles that you can apply to TextBox, Label, CheckBox, and ReportInfo controls on ActiveReports. Double-click one to open it. Each reportstyle contains a set of values for each of the standard style names:
  - Normal
  - Heading1
  - Heading2
  - Heading3
  - DetailRecord
  - ReportTitle

  When you select one of these style names on a report control, ActiveReports retrieves the style values, such as font size and color, from the specified style sheet when it runs the report.

  For more information on creating your own style sheets, see Use External Style Sheets.

- Reports: Two reports, **CategoryReport** and **ProductsReport**, are included in this sample so that you can apply styles in different ways. Open one of the reports, and select the TextBox and Label controls on it to see which style is used for each.
- StyleSheetsForm: The form in this project features radio buttons for choosing the report and style you want, a **Choose** button that opens a standard Windows **Open** dialog where you can select a reportstyle, and a **Run report** button that runs the selected report, applies the selected reportstyle, and displays the results in the ActiveReports viewer control below.
  To see how all of this works, right-click the form and select **View Code**.

  Choose Button Click Event

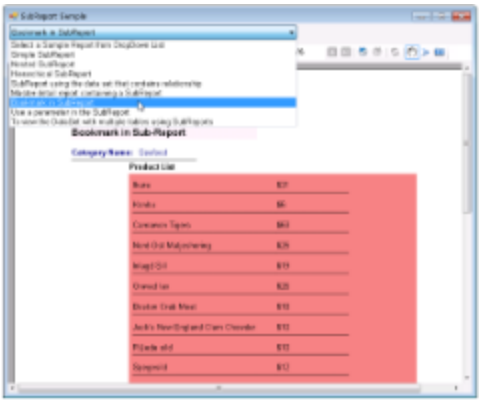
  This event contains code that sets up an Open dialog that shows only ***.reportstyle** files, and passes the selected reportstyle path and file name string to the externalStyleSheet variable.

  Run Report Button Click Event

  This event contains code that creates an empty SectionReport object, assigns the selected report to it, and assigns a path and file name string to the styleSheet variable. It then assigns the style sheet to the report using the **LoadStyles(styleSheet)** method, runs the report, and displays it in the viewer.

Sub Report

The SubReports sample demonstrates how SubReport control can be used to generate nested and hierarchical reports.



Sample Location

Visual Basic.NET

```
..\Samples14\API\Section\SubReport\VB.NET
```

C#

```
..\Samples14\API\Section\SubReport\C#
```

Details

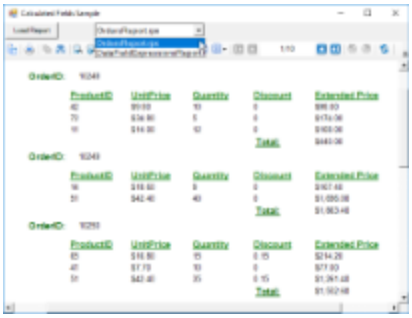
When you run this sample, the blank Viewer form appears, with the drop-down list of the sample reports on the top of the form. Select the report from the drop-down list to have it displayed in the Viewer control. You can select from the following options.

- **Simple SubReport** - the basic sample report that demonstrates how to embed a report into another report. On selecting this report, rptSimpleMain report is displayed. The Detail section of this report contains the bound Textbox control to display the Category Name information and the Subreport control to display data from rptSimpleSub.
- **Nested SubReport** - the report demonstrates how to nest subreports to display main, child, and grandchild levels in a report. It uses rptNestedParent, rptNestedChildMain, and rptNestedChildSub reports.
- **Hierarchical SubReport** - the main report dataset with the SHAPE statement defines the hierarchical structure of the report that uses a subreport. It uses rptHierarchicalMain and rptHierarchicalSub reports.
- **SubReport using the data set that contains relationship** - the main report having dataset with the relation that is defined in code, in the **DataSet.Relations** property of the main rptDSRelationParent report. It uses rptDSRelationParent, rptDSRelationChildMain, and rptDSRelationChildSub reports.
- **Master-detail report containing a SubReport** - the sample report that demonstrates how to create a master detail report that uses a subreport. It uses rptMasterMain and rptMasterSub reports.
- **Bookmark in SubReport** - the sample report that uses bookmarks from the subreport. It uses rptBookmarkMain and rptBookmarkSub reports.
- **Use a parameter in the SubReport** - the sample report demonstrates how to set up a parameter in the data source of the subreport. See rptParamMain and rptParamSub for details.
- **To view the Dataset with multiple tables using SubReports** - the sample report with the dataset that contains multiple data tables. The main report uses subreports to output multiple tables in a single report. See rptUnboundDSMain and rptUnboundDSSub for details.

> **Note:** To run this sample, you must have access to the **Nwind.mdb**. The NWIND.mdb file can be downloaded from GitHub: ..\Samples14\Data\NWIND.mdb.

Summary

This sample shows how to use calculated fields and data field expressions for simple calculations in a report.



Sample Location

Visual Basic.NET

```
..\Samples14\API\Section\Summary\VB.NET
```

C#

```
..\Samples14\API\Section\Summary\C#
```

Details

When you run this sample, The Viewer control with the **Select Report** drop-down list is displayed. There, you can select one of the two reports - **OrdersReport** or **DataFieldExpressionsReport**, and click the **Load Report** button to display the report in the Viewer.

The OrdersReport shows how to use calculated fields, where the field values are calculated in code. A custom field is added to the Fields collection in the DataInitialize event and the field value is calculated in the FetchData event.

The DataFieldExpressionsReport demonstrates the use of data field expressions for simple calculations within the same section of the unbound report using known Fields collection values. These data field expressions cannot be used with the built in summary functions of ActiveReports 14.

> **Note:** To run this sample, you must have access to the **Nwind.mdb**. The NWIND.mdb file can be downloaded from GitHub: ..\Samples14\Data\NWIND.mdb.

The sample consists of the **StartForm** and two reports - **OrdersReport** and **DataFieldExpressionsReport**.

- StartForm: The **Viewer** control has the **Dock** property set to **Fill**. This ensures that the viewer resizes along with the form at run time. Right-click the form and select **View Code** to see the code used to run the report and display it in the viewer.
- OrdersReport: The report shows the ProductID, UnitPrice, Quantity, Discount, Extended Price and Total value for each OrderID. The **Extended Price** value is a calculated field that displays the result of the formula specified in FetchData event.
  The OrdersReport uses a GroupHeader section, a Detail section and a GroupFooter section as well as a Label in the PageFooter section to display data.

  > **Note:** Except for the Detail section, all sections come in header and footer pairs. Unused sections have their

> **Height** properties set to **0** and their **Visible** properties set to **False**.

ghOrderID section

This group header section has the **DataField** property set to OrderID. This setting, along with data sorted by the same field, displays a report grouped by OrderID. The section contains one bound TextBox control to display the OrderID at the beginning of each group.

Detail section

The Detail section of this report contains 5 bound TextBox controls that render for each row of data of the OrderID.

gfOrderID section

This group footer section displays total of the gfOrderID data in TextBox controls that have values passed in code, or are bound to fields from the report's Fields collection using the DataField property. The total extended price for the OrderID is summarized using the following properties:
SummaryFunc: Sum (the default value)
Adds values rather than counting or averaging them.
SummaryGroup: ghOrderID
Summarizes the values that fall within the current OrderID group.
SummaryRunning: Group
Calculates a running summary (each value is the sum of the current value and all preceding values) within the same group level.
SummaryType: SubTotal
Summarizes the current group rather than a page or report total.

PageFooter section

The page footer section contains a static Label control that prints at the bottom of each page and contains the note on the number of pages in the report.

- DataFieldExpressionsReport: The DataFieldExpressionsReport displays data, using the Fields collection from the OrderDetail class. The report data under **ExtendedPrice** is calculated by the data field expression, specified in the **DataField** property of the **txtExtendedPrice** TextBox at the design time. The DataFieldExpressionsReport is an unbound report that uses the field values from the OrderDetail class for displaying the report data.

PageHeader section

The PageHeader section contains one Label control with the note text and four labels with the names of the report data fields.

Detail section

The Detail section contains four textboxes that use the Fields collection values to display the report data. The **DataField** property of the **txtExtendedPrice** textbox in the Detail section demonstrates how to format your data field expression.

OrderDetail class

The OrderDetail class contains data that is used in the fields of the report. When the report is run, the values of these fields are used to display data of the report. The field values are bound to the fields in the **DataInitilize** event and the data is bound to the field values in the **FetchData** event of the DataFieldExpressionsReport.

Data Binding

The samples in the Data Binding folder demonstrate data binding with various data providers separately for Page/RDL and Section reports.

- Page and RDL Reports
- Section Reports

Page and RDL Reports

This section contains:

CSV Data Source
: This sample demonstrates how to connect to a CSV data source.

DataSet DataSource
: This sample demonstrates how to use a dataset as a data source for a report.

Json Data Source
: This sample demonstrates how to use the Json data provider at run time and add a web service for authentication.

Object Data Source
: This sample demonstrates how to use Object provider for binding a report.

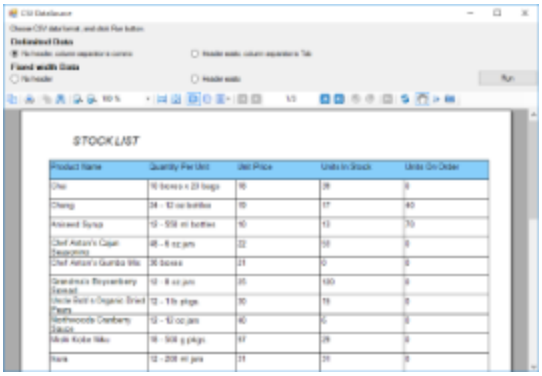
OData Data Source
: This sample demonstrates how to use OData EndPoint for binding a report.

OleDb Data Source
: This sample demonstrates how to connect to an OleDb data source at run time and pass data to the report using LocateDataSource event.

XML Data Source
: This sample demonstrates how to connect to a XML data source at run time and pass data to the report using LocateDataSource event.

CSV Data Source

The CSV Data Source sample demonstrates how to use the CSV data provider in a Page report.



Sample Location

Visual Basic.NET

```
..\Samples14\DataBinding\PageAndRDL\CSVDataSource\VB.NET
```

C#

```
..\Samples14\DataBinding\PageAndRDL\CSVDataSource\C#
```

Run-Time Features

When you run this sample, the **CSV DataSource** window appears. Select a radio button to specify the data format of the CSV file to use for the data source, and click the Run button to show the report in the viewer. You can choose from the following CSV data formats:

- Delimited Data
  - No header, column separator is comma
  - Header exists, column separator is Tab
- Fixed width Data
  - No header
  - Header exists

Project Details

MainForm

This is the main form that appears when you run this sample. It uses the ActiveReports Viewer control to display the report at run time, and radio buttons to select the data source settings.

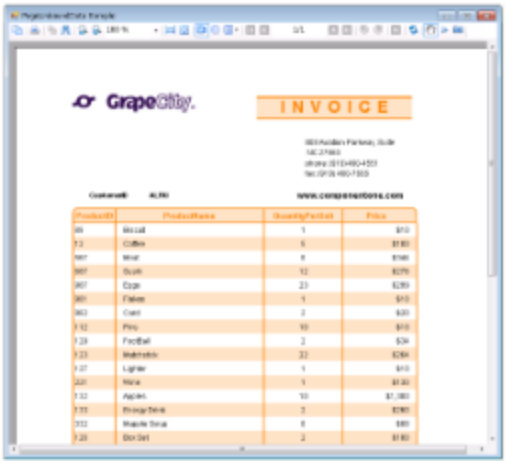
Right-click the form and select **View Code** to see how to set the data source settings in the connection string, and how to show the report at run time.

StockList.rdlx

This is the report that gets displayed in the Viewer at run time. The report is bound to the StockList dataset of the CSV data provider and uses a Table data region and TextBox controls to display the stock list. The stock list is grouped by CustomerID value.

DataSet DataSource

The DataSetDataSource sample demonstrates how to connect a page report to an unbound data source at run time, using the DataSet provider with the LocateDataSource event. The reporting engine raises the LocateDataSource event when it needs input on the data to use.



Sample Location

Visual Basic.NET

```
..\Samples14\DataBinding\PageAndRDL\DataSetDataSource\VB.NET
```

C#

```
..\Samples14\DataBinding\PageAndRDL\DataSetDataSource\C#
```

Run-Time Features

When you run this sample, the Invoice2.rdlx report is displayed in the Viewer control. The report displays the Invoice form with the list of products along with the product ID, quantity and price of the products.

The report connects to an unbound data source at run time using the LocateDataSource event and the DataSet provider.

Project Details

Invoice.rdlx

In the Report Data Source dialog, the type of the report data source is set to DataSetProvider and the ConnectionString is left blank. In the DataSet dialog, data fields used on the report are added to the DataSet on the Fields page.

This report uses the Table, TextBox, Label and Shape controls to create an Invoice layout for displaying the customer transactions. The Container control at the bottom of the report contains a label, a textbox and line control. The textbox uses the Sum function to display the sum of the values returned by the expression indicated in the Value property.

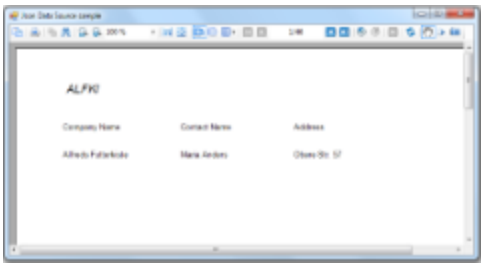
ViewerForm

This is the main form that appears when you run this sample. It uses the ActiveReports Viewer control to display the report at run time. The code-behind the form contains the code with the LocateDataSource event that connects the Invoice2.rdlx report to unbound data and the code that populates fields for the data table.

Json Data Source

The Json Data Source sample demonstrates how to use the Json data provider. The sample uses a web service that requires authentication to access the Json data source at run time.

You must have IIS Express installed on your machine for the Json Data Source sample to run.



Sample Location

Visual Basic.NET

```
..\Samples14\DataBinding\PageAndRDL\JsonDataSource\VB.NET
```

C#

```
..\Samples14\DataBinding\PageAndRDL\JsonDataSource\C#
```

Run-Time Features

The sample consists of two projects: the Windows Application project with a report, MainForm, and DataLayer class that provides report data; and the Web Application project with a web service that provides access authentication for the report data at run time.

When you run this sample, you will see the MainForm appear. The MainForm contains the ActiveReports **Viewer** that displays a report with a list of Customers from the Json data provider.

> **Note:** To run this sample, you must have access to the **customers.json** data file. The customers.json file can be downloaded from GitHub: ..\Samples14\Data\customers.json.

Project Details

JsonDataSource

This is the Windows Application project that contains the MainForm, the sample page report, and the DataLayer file with the report data.

DataLayer

This file is an internal class that contains code to create the data connection. It provides interaction with the server containing Json data using HTTP, including the access credentials.

MainForm

This is the main form that appears when you run this sample. It uses the ActiveReports Viewer control to display the report at run time.

Right-click the form and select **View Code** to see how to load and show the report at run time.

testReport.rdlx

This is the report that gets displayed in the Viewer at run time. The report contains the embedded Json schema as the data source.

It uses the Table data region to display data from the Json data source, the **Customers** sample data file.

WebService

This is the Web Application project that contains the web service used to retrieve data from the Json data source for the sample report at run time.

BasicAuthHttpModule

This file is the public class that provides access authentication when the report connects to the Json data source at run time.

Service.asmx

This is the web service required to access the Json data provider.

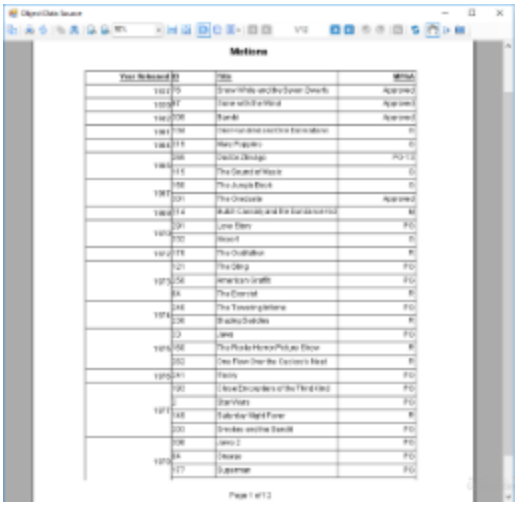
Web.config

This is the web configuration file for configuring your web application.

Object Data Source

The Object Data Source sample demonstrates how to use Object provider for binding a report that contains a subreport.

- **Sample Location**
- **Run-Time Features**
- **Project Details**



Sample Location

Visual Basic.NET

```
..\Samples14\DataBinding\PageAndRDL\ObjectDataSource\VB.NET
```

C#

```
..\Samples14\DataBinding\PageAndRDL\ObjectDataSource\C#
```

Run-Time Features

When you run this sample, the MainForm with the ActiveReports Viewer appears. The viewer displays the report where **YearReleased** column is a part of the main report (ObjectsReport.rdlx) whereas other columns such as **ID**, **Title** and **MPAA** are part of the subreport (SubObjectsReport.rdlx).

Project Details

DataLayer

This file is an internal class that contains code to provide data for the report.

MainForm

This is the main form that appears when you run this sample. It uses the ActiveReports Viewer control to display the report at run time.

Right-click the form and select **View Code** to see how to load and show the report at run time.

ObjectsReport.rdlx

This is the report that gets displayed in the Viewer at run time.

The report uses the header with the Textbox to display the report heading and the footer with a Textbox to display the page number information. The body of the report has the Table data region to display data where only first column is obtained from the Objects data provider. For that, the Textbox controls of the Table are bound to the Objects data source in the **Value** property. For rest three columns, subreport control is used.

SubObjectsReport.rdlx

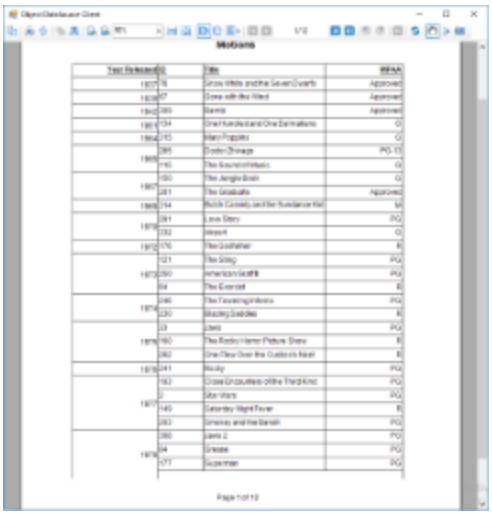
This is the subreport that is placed on Table data region of the main report.

The report has a Table data region to display data obtained from the Objects data provider. Here also, the Textbox controls of the Table are bound to the Objects data source in the **Value** property.

OData Data Source

The OData DataSource sample demonstrates how to use OData EndPoint for binding a report.

- **Sample Location**
- **Run-Time Features**
- **Project Details**



Sample Location

Visual Basic.NET

```
..\Samples14\DataBinding\PageAndRDL\ODataDataSource\VB.NET
```

C#

```
..\Samples14\DataBinding\PageAndRDL\ODataDataSource\C#
```

Run-Time Features

The ODataDataSource sample needs a running ODataEndPoint to obtain data. To run the sample, please perform the following steps:

1. Right-click solution in **Solution Explorer** and select **Properties**.
2. Select **Multiple startup projects** radio button and Start actions in **ODataEndPoint** and **ObjectDataSourceClient**, or **ODataEndPoint** and **JsonDataSourceClient** projects in Startup Project tab.
3. Run the sample again.

Project Details

The sample consists of JsonDataSourceClient and ObjectDataSourceClient projects to render data and ODataEndPoint to query data.

JsonDataSourceClient

This folder contains the DataLayer, Program and Service classes required for the data connection.

The MainForm is the form that appears when you run this sample if you have previously selected **ODataEndPoint** and **JsonDataSourceClient** projects as startup projects in the sample **Properties**.

ObjectDataSourceClient

This folder contains the DataLayer, Program and Service classes required for the data connection.

The MainForm is the form that appears when you run this sample if you have previously selected **ODataEndPoint** and **ObjectDataSourceClient** projects as startup projects in the sample **Properties.**

The **Models** subfolder contains Movie and Year classes.

ODataEndPoint

This folder contains the **AppData** and **AppStart** subfolders required to run the application.

The **Controllers** subfolder contains the **MoviesController** and **CustomersController** files. The **MoviesController** handles the user interaction and returns the main view. The **CustomersController** handles the customer details information that is displayed when a customer is selected.

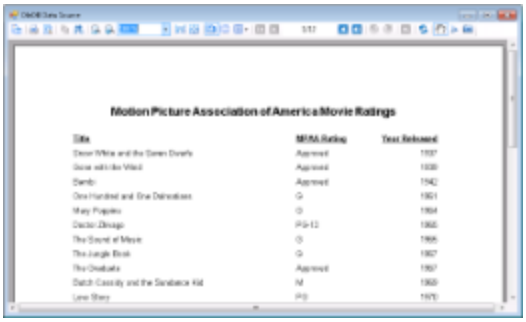
The **Models** subfolder contains the Customer and Movie classes providing data for the report.

Global.asax is the default class that sets global URL routing values for this web application.

Web.config is the configuration file that contains the httpHandlers that allow ActiveReports to process this web application. Note that you need to manually update version information here when you update your version of ActiveReports.

OleDb Data Source

The OleDB Data Source sample demonstrates how to use the OleDb data provider for binding a report to data.



Sample Location

Visual Basic.NET

```
..\Samples14\DataBinding\PageAndRDL\OleDbDataSource\VB.NET
```

C#

```
..\Samples14\DataBinding\PageAndRDL\OleDbDataSource\C#
```

Run-Time Features

When you run this sample, the MainForm with the ActiveReports Viewer appears. The Viewer displays the report with the list of movies, their ratings and the release year information.

> **Note:** To run this sample, you must have access to the **Reels.mdb**. The Reels.mdb file can be downloaded from GitHub: ..\Samples14\Data\Reels.mdb.

Project Details

DataLayer

This file is an internal class that contains code to create a data connection. It creates the OleDb data reader to read data from the Reels database and add it to an array to provide data for the report.

MainForm

This is the main form that appears when you run this sample. It uses the ActiveReports Viewer control to display the report at run time.

Right-click the form and select **View Code** to see how to load and show the report at run time.

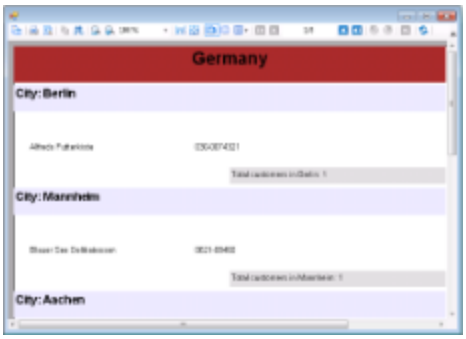
OleDBReport.rdlx

This is the report that gets displayed in the Viewer at run time.

The report uses the header with the Textbox to display the report heading and the footer with the Textbox to display the page number information. The body of the report has the Table data region to display data obtained from the OleDb data provider. For that, the Textbox controls of the Table are bound to the OleDB data source in the **Value** property.

Xml Data Source

The XML Data Source sample demonstrates how to use the XML data provider for supplying data to the report.



Sample Location

Visual Basic.NET

```
..\Samples14\DataBinding\PageAndRDL\XmlDataSource\VB.NET
```

C#

```
..\Samples14\DataBinding\PageAndRDL\XmlDataSource\C#
```

Run-Time Features

When you run this sample, you will see the MainForm appear. The MainForm contains the ActiveReports **Viewer** that displays a report with data from the xml data provider.

Project Details

BandedListXML.rdlx

This is the main report that gets displayed in the Viewer at run time. It uses the BandedList and Textbox controls, and the Subreport control inside the BandedList data region to display data.

The BandedList data region uses two groups to group the report data by the fields **City** and **Country**.

The Subreport control, placed in the GroupFooter section of the BandedList, displays the **CountrySales** report.

CountrySales.rdlx

This is the report that gets displayed by the Subreport control of the BandedListXML report.

It uses the Chart data region to display data. The **Chart Type** property is set to **Doughnut (Pie Exploded Doughnut)**, which shows the analysis of companies sales amount for different countries.

Section Reports

This section contains:

Bound Data
: Demonstrates binding to ADO.NET Data objects.

IList Binding
: Demonstrates creating a custom collection that stores data from the database in the List. The custom collection is displayed by binding data to the DataGridView control by using the DataSource property of this control.

LINQ
: The LINQ sample demonstrates how to use LINQ in an ActiveReports report.

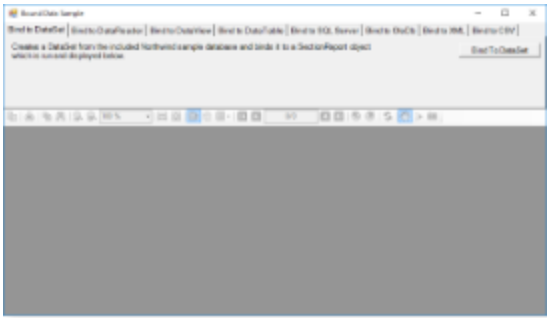
Unbound Data
: The Unbound Data sample demonstrates how to create a dataset for a section report and use the FetchData event to populate the Fields collection to display the report unbound data.

XML
: This sample demonstrates how to create a report with XML data, using a SubReport or using the XML hierarchical structure.

Bound Data

The Bound Data sample demonstrates various ways to bind data in a section report.

When you run the sample, the Viewer control displays the form with eight tabs, each with a different data binding technique. Click to select a tab, and then click the **Bind To** button to create the report.



Sample Location

Visual Basic.NET

```
..\Samples14\DataBinding\Section\BoundData\VB.NET
```

C#

```
..\Samples14\DataBinding\Section\BoundData\C#
```

Run-Time Features

The top panel of the MainForm is composed of eight tabs:

Bind to DataSet
Creates a DataSet from the sample database and binds it to a SectionReport object.

Bind to DataReader
Creates a DataReader from the sample database and binds it to a SectionReport object.

Bind to DataView
Creates a DataView from the sample database and binds it to a SectionReport object. This tab contains a ComboBox which lets you choose the company name from the NWind database.

Bind to DataTable
Creates a DataTable from the sample database and binds it to a SectionReport object.

Bind to SQL Server
Creates a SQL Server DataSource from a SQL server instance and binds it to a SectionReport object. The ComboBox present in this tab lets you populate the dropdown list with the existing SQL servers on the network.

Bind to OleDb
Creates an Oledb DataSource and binds it to a SectionReport object.

Bind to XML
Creates a XML DataSource from a file and binds it to a SectionReport object. The XML tab also features a **Generate XML** button that generates a DataSet and saves it as an XML data file. The generated file is then used as a data source for the report.

Bind to CSV
Creates a CSV DataSource from a file and binds it to a SectionReport object. You can select the data type of the file from the following options:

- Delimited Data (with or without header)
- Fixed width Data (with or without header)

Project Details

MainForm

The MainForm uses the ActiveReports **Viewer** control in the bottom section of the form, and a panel docked to the top contains tabs, each with a different data binding technique.

Click to select a tab, and then double-click the button on the tab to jump to the button's **Click** event in the code.

Invoice Report

The Invoice report uses three GroupHeader sections, a Detail section and a GroupFooter section as well as a label in the PageFooter section to display data.

> **Note:** Except for the Detail section, all sections come in header and footer pairs. Unused sections have their **Height**

> properties set to **0** and their **Visible** properties set to **False**.

ghOrderHeader

The **DataField** property of this section is set to **OrderID**. This setting, in conjunction with data ordered by the OrderID field, causes the report to print all of the information for one order ID value, including all of the related details and footers, before moving on to the next order ID. For more information on grouping, see Add Groups.

This section also contains a Picture control, a number of Label controls, and two bound TextBox controls. The TextBoxes are bound using the **DataField** property in the Properties window, and the date is formatted using the **OutputFormat** property.

ghOrderID

The **DataField** property of this section is also set to **OrderID**. This allows subtotal summary functions in the related GFOrderID section to calculate properly.

This section contains a number of labels and bound text boxes, as well as two **Line** controls.

ghTableHeader

This section contains only labels for the data to follow in the Detail section.

Detail

This section contains bound TextBox controls. These TextBoxes render once for each row of data found in the current OrderID before the report moves on to the GroupFooter sections.

GFOrderID

The **NewPage** property of this section is set to **After**. This causes the report to break to a new page and generate a new invoice after this section prints its subtotals.

This section contains several labels and several TextBoxes. Two of the TextBox controls use the following properties to summarize the detail data: **SummaryFunc**, **SummaryGroup**, and **SummaryType**. For more information, Create a Summary Report.

The **Total** TextBox does not use the DataField property or any of the summary properties, or even any code. To find the functionality of this TextBox, in design view, click the **Script** tab at the bottom of the report.

PageFooter

This section has one simple Label control. For more information about report sections and the order in which they print, see Section Report Structure and Section Report Events.

ProductList Report

The ProductList report uses the Header and Detail sections to display data.

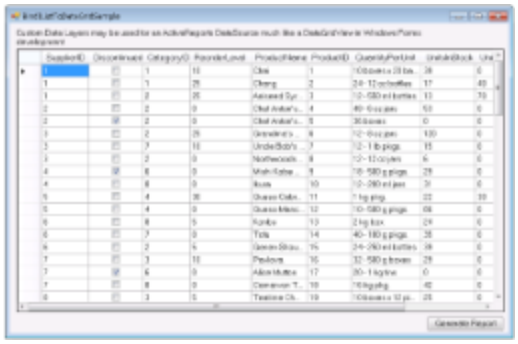
The Header section contains a number of Label controls to display column names for the product list.

The Detail section contains four TextBox controls to fetch the product data.

IList Binding

The IList Binding sample uses **CollectionBase** class, with implementation of IList interface to create a **ProductCollection** class which gets populated from the Products table of the NWind database. The created ProductCollection is used as a database for binding data to the DataGridView control.The data from ProductCollection class gets displayed using the DataSource property of DataGridView control. On clicking the Generate Report button, the ProductCollection class is again used to display the data of the generated report in a Viewer control. Similarly, you can display a report by binding to the DataSource property of a report.

Data from ProductCollection class displayed in DataGridView control



Generated report displayed in Viewer control

Sample Location

Visual Basic.NET

```
..\Samples14\DataBinding\Section\IListBinding\VB.NET
```

C#

```
..\Samples14\DataBinding\Section\IListBinding\C#
```

Run-Time Features

When you run this sample, DataGridView, which is a standard Windows Forms control displays the custom collection. To display a report with the bound custom collection, click the **Generate Report** button.

> **Note:** To run this sample, you must have access to the **Nwind.mdb**. The NWIND.mdb file can be downloaded from GitHub: ..\Samples14\Data\NWIND.mdb.

Project Details

The IList Binding sample consists of two projects: **IListBinding** and **IListBinding.DataLayer**.

IListBinding Project

BindIListToDataGridSample

This form contains a DataGridView control, a Label control and a Generate Report button. It displays output results by binding data of a custom collection to the DataGridView control. On clicking the **Generate Report** button, ViewerForm displays a report bound to this custom collection.

IlistReportSample report

The report uses the ReportHeader, GroupHeader1 and Detail sections for the report output.

ReportHeader section
The ReportHeader section contains a Label that displays the title of the report.

GroupHeader1 section
The GroupHeader1 section contains nine Label controls that define the layout of the report data.

Detail section
The Detail section contains TextBox controls to display the report data. Following settings have been performed to enhance the appearance of the report output.

- Change the background color of alternate rows
  Use the **BackColor** property of the Detail section (set in the Format event of the Detail section) to change the background color of each row for better visibility of the table.
- Change the background color for selected rows
  Use the **BackColor** property of the Detail section (set in the Format event of the Detail section) to change the background color of selected rows. The background color changes when Reorder Level is below Ordered Units.

ViewerForm

Setting the **Dock** property of the Viewer control to **Fill** ensures that the viewer resizes along with the form at run time. Right-click the form and select **View Code** to see the code used to run the report and display it in the viewer.

IListBinding.DataLayer project

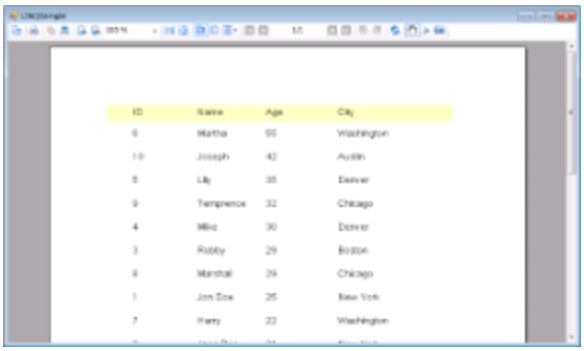
DataProvider Class
Implements the connection to the data base.

Product Class
Defines custom collection class.

ProductCollection Class
Implements the CollectionBase class to create a collection of Product class. The list of this collection stores data from the Products table.

LINQ

The LINQ sample demonstrates how to use LINQ in an ActiveReports report.



Sample Location

Visual Basic.NET

```
..\Samples14\DataBinding\Section\LINQ\VB.NET
```

C#

```
..\Samples14\DataBinding\Section\LINQ\C#
```

Run-Time Features

This sample uses a LINQ Query to sort recordsets in descending order of age. The resultant recordsets are converted to an IList and used as a datasource for the report which is displayed in Viewer control.

Project Details

ViewerForm

Displays the report output results. ToList method is set in **DataSource** property of the report to extract objects that use LINQ.

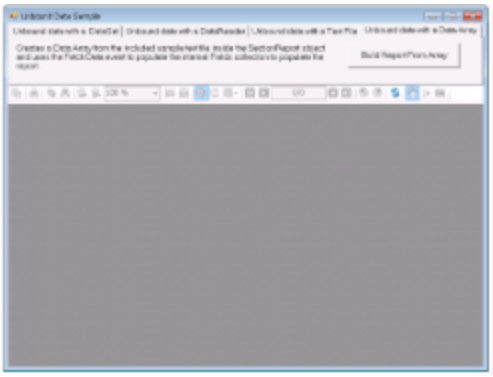
rptLINQtoObject report

The report DataSource is a list of Person constructor created from generic class. Creates a query to sort in descending order of Age.

Unbound Data

The Unbound Data sample demonstrates how to create a dataset for a section report and use the FetchData event to populate the Fields collection to display the report unbound data.

When you run the sample, the Viewer control displays the form with four tabs, each with a different dataset binding technique. Click to select a tab, and then click the **Build Report From** button to create the report with unbound data.



Sample Location

Visual Basic.NET

```
..\Samples14\DataBinding\Section\UnboundData\VB.NET
```

C#

```
..\Samples14\DataBinding\Section\UnboundData\C#
```

Run-Time Features

- **Unbound data with a DataSet**
  Creates a data set from the included Northwind sample database inside the SectionReport object and uses the FetchData event to populate the internal Fields collection to display the report data.
- **Unbound data with a DataReader**
  Creates a data reader from the included Northwind sample database inside the SectionReport object and uses the FetchData event to populate the internal Fields collection to display the report data.
- **Unbound data with a Text File**
  Sets the Invoice.txt file as a datasource for the SectionReport object and uses the FetchData event to populate the internal Fields collection to display the report data.
- **Unbound data with a Data Array**
  Creates a data array from the included sample text file inside the SectionReport object and uses the FetchData event to populate the internal Fields collection to display the report data.

Project Details

MainForm

This is the main form of the sample that uses the ActiveReports **Viewer** control in the bottom section of the form, and a panel docked to the top contains four tabs, each with a different data binding technique. Click to select a tab, and then click the button on the tab to display the report with unbound data.

UnboundDAInvoice report

The Invoice report for the **Unbound data with a Data Array** option. The report consists of the page header, group header, detail, group footer and page footer sections. The detail section contains information on the order details, the group header provides grouping data functions by using its **DataField** property.

For the details on the Invoice report, see the Bound Data Sample topic.

UnboundDRInvoice report

The Invoice report for the **Unbound data with a DataReader** option. The report consists of the page header, group header, detail, group footer and page footer sections. The detail section contains information on the order details, the group header provides grouping data functions by using its **DataField** property.

For the details on the Invoice report, see the Bound Data Sample topic.

UnboundDSInvoice report

The Invoice report for the **Unbound data with a DataSet** option. The report consists of the page header, group header, detail, group footer and page footer sections. The detail section contains information on the order details, the group header provides grouping data functions by using its **DataField** property.

For the details on the Invoice report, see the Bound Data Sample topic.

UnboundTFInvoice report

The Invoice report for the **Unbound data with a Text File** option. The report consists of the page header, group header, detail, group footer and page footer sections. The detail section contains information on the order details, the group header provides grouping data functions by using its **DataField** property.

For the details on the Invoice report, see the Bound Data Sample topic.

XML

The XML sample displays customer order list using the XML data source. The sample demonstrates how to create a report with XML data, using a SubReport or using the XML hierarchical structure.

Sample Location

Visual Basic.NET

```
..\Samples14\DataBinding\Section\XML\VB.NET
```

C#

```
..\Samples14\DataBinding\Section\XML\C#
```

Run-Time Features

When you run the sample, you will be asked to select between the following.

Run Customers XML Report
Demonstrates a two level Master/Detail XML Data Source. Uses a SubReport to print the orders of each customer.
By selecting any of the radio buttons - **Show All Data**, **Show Report where ID = 2301**, or **Show All Data with an E-Mail address**, you can change the creation option of the generated report.

Customer Orders (Leveled)
Displays customer's orders by using the XML hierarchical structure. The Master/Detail is implemented using the XML

hierarchy XPath selection patterns without any SubReports.

Project Details

StartForm

This is the main form of the sample. You see this form after you run the project and where you can select how to display XML data.

- Run Customers XML Report
  Displays the customers order list by using SubReports.To bind the CustomersOrders report to the XML data source, the valid XPath expression is entered in the **RecordsetPattern** field on the XML tab of the **Report Data Source** dialog.

- Customer Orders (Leveled)
  Displays customer's orders by using the XML hierarchical structure, which is demonstrated by the OrdersLeveled report. To bind this report to XML data, the path to the XML file is entered in the **File URL** field on the XML tab of the **Report Data Source** dialog and also the XML hierarchical structure like "../../@email" is specified in each field.

CustomersOrders report

This report embeds the srptOrders SubReport. Following settings are performed in this report.

- Embed the SubReport control
  Places the SubReport control in the Detail section and connects it to the Orders SubReport for displaying the orders list.

OrderItems SubReport

This report binds to the Subreport control of Orders report.

Orders SubReport

This is the report with a SubReport control that is bound to the OrderItems report. Following settings are performed in this report.

- Embed the SubReport control
  Places the SubReport control in the Detail section and connects it to the OrderItems report for displaying the list of orders.
  The Report property of the Subreport control is specified in the Orders_ReportStart event.
- Set the XML data source included in the SubReport
  Indicates a method to retrieve the record set using the NodeList property instead of the RecordsetPattern property of the XML data source on OrderItems report at design time.
  The NodeList property of the XML data source is set in the Detail_Format event.

OrdersLeveled report

Displays the list of customer's orders. Following settings are performed in this report.

- Groups data
  Groups data using the XPath pattern that represents the `hierarchical` structure of XML. DataField property of ghCustomers section and ghOrders section is set at design time.

ViewerForm

The Viewer control has its Dock property set to Fill. This ensures that the viewer resizes along with the form at run time. Right-click the form and select View Code to see the code used to run the report and display it in the viewer.

Designer Pro

The samples in the Designer Pro folder demonstrate features provided with the ActiveReports professional edition.

Map
: This sample demonstrates how to work with Map control in ActiveReports.

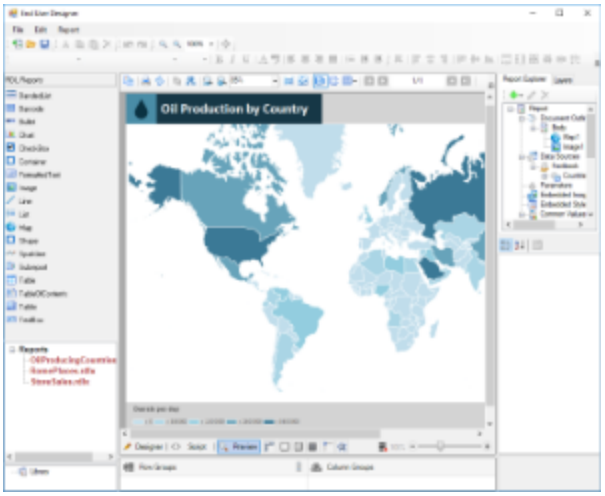
End User Designer
: This sample demonstrates a custom end-user report designer that can be integrated in your applications to allow users to modify report layouts.

Reports Gallery
: This sample demonstrates customizing End User Designer application to display a list of categorized reports.

Table of Contents
: This sample demonstrates how to use TableofContents control in ActiveReports.

Map

The Map sample demonstrates the basic functioning of the Map control with the help of four reports that explain the different features of the control. This sample is part of the ActiveReports Professional Edition.



Sample Location

Visual Basic.NET

```
..\Samples14\DesignerPro\Map\VB.NET
```

C#

```
..\Samples14\DesignerPro\Map\C#
```

Details

When you run this sample, the End User Designer shows a list of .rdlx reports at the bottom left of the form. Expand the

Reports node to view reports under it and double-click a report to load it into the designer.

Report Form

This is the main form that appears when you run the sample. This form uses the ToolStripPanel, ToolStripContentPanel, Designer, Toolbox, ReportExplorer and PropertyGrid controls to create a customized ReportDesigner.
Right-click the form and select **View Code** to see how to set up the designer and create a blank page report. It also contains code that adds the reports to the TreeView control, loads a report into the Designer when the report is double-clicked, and checks for any modifications that have been made to the report in the designer.

Reports Folder

OilProducingCountries.rdlx: This report uses the FactBook shared data source connection to provide data.

It contains a Map control that visually displays the oil production in different countries of the world on a virtual earth background. The map control uses the color rule set on a polygon layer to differentiate parts of world according to their oil production capacity. These colors are defined using a color rule which is described in the legend at run time.

RomePlaces.rdlx: This report contains a Map control that visually displays famous places in Rome.

The map control uses Google maps in a Tile layer to provide a virtual earth background and a Point layer to plot famous places on the map using image markers. Clicking the image marker opens the web page for the selected place in Wikipedia.

StoreSales.rdlx: This report contains a Map control that visually displays the sales of different stores in the US.

The Map control uses the built-in USA map template. The polygon layer defines the country and state boundaries while the point layer is used to plot store locations. The Point layer uses the marker size rule to differentiate stores according to their profits. The different sized markers used for plotting stores are defined in a legend that appears on the map at run time. The point layer also uses the drill-through link (the Action is set to Jump to report) that opens a specific store report (StoreReport2.rdlx) when the marker is clicked.

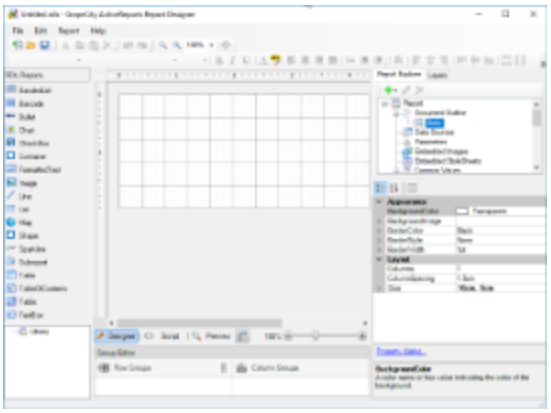
StoreReport2.rdlx: This report uses the **Reels** shared data source connection to provide data.

It opens on clicking a specific store location that is indicated with a marker. The Table data region in the report uses a drill-down link to display the list of employees in the selected store. The report uses the chart data region to display the sales for the store. The Reels logo that appears on the report in the Image control is embedded within the Reels theme.

End User Designer

The EndUserDesigner Sample demonstrates how to set up a custom end-user report designer using the Designer, ReportExplorer, Layer List, ToolBox, ReportsLibrary, and GroupEditor controls. This Sample is part of the ActiveReports Professional Edition.

> ⚠ Type your Caution Box content here.



Sample Location

Visual Basic.NET

```
..\Samples14\DesignerPro\EndUserDesigner\VB.NET
```

C#

```
..\Samples14\DesignerPro\EndUserDesigner\C#
```

Run-Time Features

When you run the sample, the End User Designer appears in the Viewer control. This report designer provides the functionality of the ActiveReports Designer and supports three types of report layouts: Page Report, RDL Report, and Section Report.

The End User Designer lets you create report layouts and edit them at design time or runtime. The Designer includes the Property Window with extensive properties for each element of the report, the Toolbox is filled with report controls, the Report Explorer with a tree view of report controls, the Reports Library displaying report parts (group of controls) in a report, and a Group Editor displaying row and column groups for Tablix data region. Page reports and RDL reports provide the Layer List in a tabbed window with the Report Explorer. The Layer List window displays a list of layers in the report along with their visibility and lock options.
See Report Parts, Layers, and Tablix for more information. For information on menu items, see ActiveReports Designer.

The project consists of following forms:

ExportForm

This is the form with the **Export** dialog for Page report, Rdl report and Section report.

A user sees the **Export** dialog under the **Preview** tab in the **File** menu > **Export**. This dialog allows to select the export type and to browse for the file location in local folders where the report is exported. See Exporting for details on the type of export formats supported in Section report, Page report, and RDL report.

Control	Name	Description
ComboBox	cmbExportFormat	The **Export Format** combo box that allows to select options for the report export type.
Button	btnOK	The **OK** button in the lower part of the ExportForm.
Button	btnCancel	The **Cancel** button in the lower part of the ExportForm.
PropertyGrid	exportPropertyGrid	Provides interface for export options of each export type.

SaveFileDialog	exportSaveFileDialog	The **Save File** dialog that allows to specify the file name for saving an exported report file.
Label	lblExport	The **Export** label in the header of the ExportForm.
Label	lblExportFormat	The **Export Format** label of the Export Format combo box.
Label	lblExportOptions	The **Export Options** label of the Export property grid.
Label	lblSelectExportTxt	The **Select Export** text that describes the purpose of the ExportForm.
SplitContainer	exportSettings SplitContainer	Represents a movable bar that divides the display area of the ExportForm into two resizable panels - the ExportForm header panel and the ExportForm panel with the Export Format combo box and the Property Grid.
SplitContainer	exportHeader SplitContainer	Represents a movable bar that divides the Export Format section consisting of the Export Format combo box with the Export Format label and the Property Grid of ExportForm.

Right-click the ExportForm in the Solution Explorer and select **View Code** to see the code implementation for the Export form.

EndUserDesigner form

This is the form with a basic end-user report designer that contains the following elements. These elements are dragged from the Visual Studio toolbox onto the form.

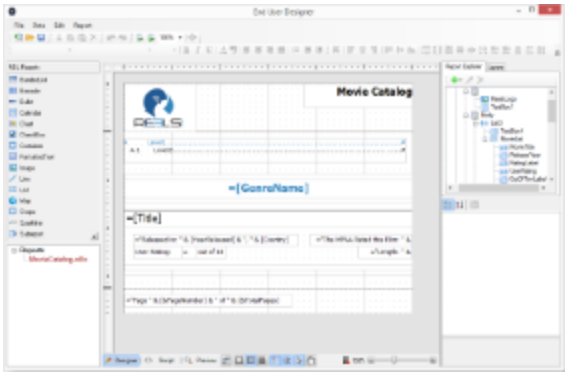
Control	Name	Description
Designer	reportDesigner	The Designer control that allows you to create and modify a report.
ReportExplorer	reportExplorer	Gives you a visual overview of the report elements in the form of a tree view where each node represents a report element.
TabControl	reportExplorerTabControl	Represents a movable bar that divides the display area of the designer into two tabs - the Report Explorer and the Layer List
PropertyGrid	reportPropertyGrid	Provides an interface for each element of the report.
Toolbox	reportToolbox	Displays all of the controls specific to the type of report that has focus.
LayerList	layerList	Represents a list of Layers in the report along with their visibility and lock options.
SplitContainer	mainContainer	Represents a movable bar that divides the display area of the Designer into two resizable panels.
SplitContainer	designerExplorerPropertyGridContainer	Represents a movable bar that divides the display area of the designer into two resizable panels -the toolbox and the toolstrip.
SplitContainer	bodyContainer	Represents a movable bar that divides the display area of the Viewer into two resizable panels.

SplitContainer	explorerPropertyGridContainer	Represents a movable bar that divides the display area of the designer into two resizable panels - the report explorer and the property grid.
ToolStripContainer	toolStripContainer	Provides a central panel on top of the Designer to hold the Toolstrip element with the menu items.
ReportsLibrary	reportsLibrary	Displays all reports and included report parts.
GroupEditor	groupEditor	Shows the row and column groups in a Tablix data region.

Right-click the EndUserDesigner form in the Solution Explorer and select **View Code** to see the code implementation for the End User Designer.

Table of Contents

The TableofContents sample demonstrates the basic features of the TableofContents control. This sample is part of the ActiveReports Professional Edition.



Sample Location

Visual Basic.NET

```
..\Samples14\DesignerPro\TableOfContents\VB.NET
```

C#

```
..\Samples14\DesignerPro\TableOfContents\C#
```

Details

When you run this sample, the End User Designer appears with a MovieCatalog.rdlx under the Reports node. The report contains a TableOfContents control which displays a list of movie titles along with their page numbers under each genre. On clicking the movie title, the details on the selected movie are displayed.

Report Form

This is the main form that appears when you run this sample. This form uses the ToolStripPanel, LayerList, TreeView, ToolStripContentPanel, Designer, Toolbox, ReportExplorer and PropertyGrid controls to create a customized ReportDesigner.

Right-click the form and select **View Code** to see how to set up the designer. It also contains code that adds the reports to the TreeView control, loads a report into the Designer when it is double-clicked, and checks for any modifications that have been made to the report in the designer.

Reports Folder

MovieCatalog.rdlx: This report uses the Reels data source connection to provide data.

The report makes use of TableOfContents, Image, TextBox, Label and List controls to display the layout of the report. TableofContents control displays an organized hierarchy of movie titles under each genre with two heading levels. Clicking the genre name or the movie title takes you to the corresponding page number that contains the details. The Reels logo that appears on the report is embedded within the Reels theme.

Desktop

The samples in Desktop folder demonstrate features of WPF and Win Viewers.

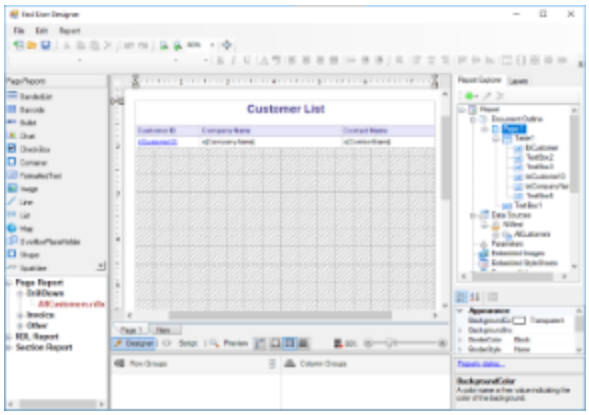
Reports Gallery
: This sample demonstrates customizing End User Designer application.

WPF Viewer
: This sample demonstrates using WPF Viewer in a WPF application.

Win Viewer
: This sample demonstrates using Win Viewer in a Windows Form application.

Reports Gallery

The Reports Gallery sample demonstrates how to customize the End User Designer application by adding the TreeView control to display a list of categorized reports. At run time, the user can double-click any report out of the three categories: Page, Rdl, and Section, and load it into the designer.



Sample Location

Visual Basic.NET

```
..\Samples14\DesignerPro\ReportsGallery\VB.NET
```

C#

```
..\Samples14\DesignerPro\ReportsGallery\C#
```

Run-Time Features

When you run this sample, the End User Designer appears with a list of report categories at the bottom left of the Form. Expand each category to view the reports under it and double-click any report to load it into the designer. You can also go to the Preview Tab to view the report.

Project Details

ReportsForm

This is the main form that appears when you run this sample. This form uses the ToolStripPanel, ToolStripContentPanel, Designer, Toolbox, ReportExplorer, TreeView, PropertyGrid and LayerList controls to create a customized unified ReportDesigner.

Right-click the Form and select **View Code** to see how to set up the designer and create a blank page report. It also contains code that attaches the Toolbox, ReportExplorer, PropertyGrid and LayerList controls to the Designer, inserts DropDown items to the ToolStripDropDownItem and sets their functions. Finally, it also contains code that adds the reports to the TreeView control, loads a report into the Designer when the report is double-clicked, and checks for any modifications that have been made to the report in the designer.

Reports folder

Reports folder consists of three subfolders - Page Report, RDL Report and Section Report, each containing a set of reports that highlight the major features of the corresponding report types.

Page Report folder

This folder contains several subfolders that illustrate the use of Page Reports in different scenarios.

DrillDown

AllCustomers report: This report uses the NWind shared data source. It uses the **Table** data region to display customer's contact details. The **Textbox** displaying the **CustomerID** field in the detail row of the Table is used to set a drill-through link to navigate to the **CustomerDetails** report.

Invoice folder

BillingInvoice report: This report uses the NWind shared data source. It showcases a billing invoice layout commonly used in convenience stores. The report mostly contains Label, TextBox and Line controls in its layout. It also includes an **EAN128FNC1** barcode due to its high reading accuracy.

Invoice1 report: This report uses the NWind shared data source. It includes a BandedList and few TextBox controls to create the Invoice layout for displaying customer transactions. Both the page and the BandedList control are grouped by the **OrderID** field. One of the text boxes in the footer section of the BandedList control uses the Sum function to display the grand total of all transactions.

Invoice2 report: This report uses the NWind shared data source. It uses a Table, few TextBoxes and Shape controls to create the Invoice layout for displaying customer transactions. The report page is grouped by the **CustomerID** field, therefore all transactions made by a customer appear together based on the ID. One of the text boxes placed inside the Container control at the bottom of the report uses the Sum function to display the grand total of all transactions.

Invoice_Grouped report: This report uses the NWind shared data source. It uses the Table, few TextBox and Label

controls to display customer transactions in an Invoice. The data is grouped on the **CustomerID** field, so that all transactions made by a customer appear together based on their ID.

Invoice_Parameters report: This report uses the NWind shared data source connection and two datasets to provide data. It is similar to the **Invoice_Grouped** report with an additional parameters feature. It uses parameters set on the **CompanyName** field to filter data on report preview.

Other

BarCode report: This report demonstrates all barcode types that are supported by ActiveReports. The barcode types are presented in the Table data region, using a single page layout. The rows of the table use alternate background colors (grey and white). At run time, the Barcode report displays one page that fits the table with all the sample barcodes.

Catalog report: This report uses the NWind shared data source. It shows a multi page layout spread over four pages in the report. The layout in **Page1** and **Page2** contains Image, Label and Textbox controls to display introductory text. The layout on **Page3** contains a List data region with TextBox controls and a Table to display product details for each product category. The List is grouped by the **CategoryID** field to filter products by their category and its FixedSize property is set to fit in excess data. The layout on **Page4** (that appears as page 9 at run time) uses Textbox, Shape and Line controls amongst others to create an Order Form, which a user is to fill manually.

CellMerging report: This report uses the NWind shared data source. This report demonstrates cell merging in Tablix data region, where cells with same values are merged automatically to avoid showing duplicate values. The row group area contains three groups that are nested in a parent/child relationship to display the row group data. The Country (parent) and City group (child) values are merged automatically to remove duplicate data values.

DeliverySlip_theme report: This report uses the **Seikyu2** shared data source. It uses the TextBox, Label, Container controls and two Table data regions to display the invoice information. The report uses two themes and has its **CollateBy** property set to **ValueIndex** to determine the order in which the report pages are rendered. Some TextBox controls on the report also use the Sum function to display the total price information for each Invoice. The page is grouped on the **EstimateID** field, so the invoices are sorted by **EstimateID**.

EmployeeSales report: This report uses the NWind shared data source. It contains the Chart and Table data regions to display sales by each employee for the year 1997. A column chart shows the graphical representation of sales by each employee while the Table lists down the exact sales figures. One of the TextBox controls on the report also uses the Sum function to display the grand total of all sales.

Enterprise Reports - Marketing Plan Data: This report displays information for the Marketing Plan. The report contains embedded XML data. Table, TextBox, Line, Image, and Shape controls are used to display data. The status of each task is represented through color coding using expressions.

IRS-W4 report: This report uses the IRS XML data source to provide data to the report. It mainly contains Textbox, CheckBox, Shape and Line controls to create the layout of a tax form used in the US.

Letter report: This report uses the NWind shared data source. It contains an Image, FormattedTextBox, Table and an OverflowPlaceholder controls to create a layout for a letter. The FormattedTextBox control uses text with HTML tags to display content. The Table displays OrderID with order dates and order amount and is linked with the OverflowPlaceholder control to display excess data on the same page. The Reels logo, displayed on the report inside the Image control, is embedded within the Reels theme and the TextBox placed below the Image control uses a Global expression to display the current date. The layout page is grouped by the **CustomerID** field to filter data on each report page according to individual customers.

MyOrdersReport: This report uses MyOrders.csv as a data source to demonstrate the native support for CSV data providers. It contains a Table data region that displays orders with data fetched from the CSV data source.

PurchaseReport: This report uses the NWind shared data source. It contains the Textbox and Table controls to display purchase details for each company. The layout page is grouped by the Company_CD field to filter data on each report page according to the company. The Textbox control placed below each table column uses the **Sum** function to display

totals. The FixedSize property of the Table is set to fit in excess data.

ReelsTablix report: This report uses the Reels shared data source. It contains the Tablix data region to display a sales report for each country, city and media type by year. The Tablix data region uses row grouping to group the data by Country, City and MediaType**,** and column grouping to group the data by Years and Quarters.

ResourceConsumptionByYear report: This report uses MostPopulatedCountriesEnergyUsageByYear shared data source. It illustrates composite charts by using a Chart data region that shows annual Natural Resource Consumption - Oil and Electricity - versus Population size. Natural Resource Consumption and Population size are plotted on the two Y axes represented by two different chart types - Column and Line. The report also contains a parameter which lets users choose the country for which they want to view the data.

SalesReport: This report uses the Reels data source connection and two datasets that fetch data through a Stored Procedure. The layout of the report uses the Chart to display sales and profit for the selected date range and the Table to display the numeric values of the same. The table also uses the **DataBar** function to plot profits graphically. The Reels logo, displayed on the report inside the Image control, is embedded within the Reels theme. This report also uses two nullable parameters to select the range of dates.

TablixSample report: This report uses the NWind shared data source. It contains the Tablix data region that displays an orders report with product category and names showing quarterly and yearly orders. The Tablix uses a nested row grouping to group the Tablix data region by CategoryName and ProductName, and nested column grouping to group the Tablix data region by Years and Quarter. The tablix body area displays the aggregate Sum for each category and the grand total of order amount.

TackSeal report: This report uses the NWind shared data source. It contains the OverflowPlaceholder control and the List data region to create a columnar display of tack seals with postal barcodes. The List data region has its **OverflowName** property set to **OverflowPlaceHolder1** and the OverflowPlaceHolder1 control has its **OverflowName** property set to **OverflowPlaceHolder2** to assure correct display of columnar data display within the report page.

RDL Report folder

This folder contains several subfolders that illustrate the use of RDL Reports in different scenarios.

Dashboard

CallCenterDashBoard report: This report uses the CallCenter shared data source. It contains uses the Bullet control to indicate when the data is approaching or past a warning range and the Sparkline controls to indicate daily trends in key pieces of performance and sales data. It also uses the Icon Set data visualizer to indicate the warning levels for key performance data.

MarketDashBoard report: This report uses the MarketData shared data source. It contains the Sparkline control to display stock price trends over the last 30 days. The Sparkline allows investors to see trends without knowing what actual values are associated with each point.

TeamList report: This report uses the FootballStatistics shared data source. It contains the List control that displays a list of team names with drill-through links to navigate to the **TeamStatisticsDashboard.rdlx** report that displays the selected team's statistics.

SalesDashboard report: This report uses the SalesResult shared data source. It consists of two datasets to display multiple Chart controls and a Tablix data region to visualize the sales performance data. This report illustrates the Galley-mode feature where all of the report contents can be previewed in a single scrollable page.

Factbook

CountryFacts report: This report uses the Factbook shared data source. It contains a List data region grouped on CountryID that groups data by **Country**. The report also contains a map image whose **Value** property is set to the

expression with the **MapCode** data field from the XML data source. It contains a hidden parameter that is used to accept the **Country ID**. This report is called in other reports by drill-through links or as a subreport, so the Country ID is passed silently. The default ID is the World. The dataset has a filter that retrieves only countries, specified by the report parameter. From each textbox with category under **Energy Production / Consumption** you can see the Image control that uses the Icon Set data visualizer to flag energy categories where consumption is greater than production.

LifeExpectancyByGdpAndMedianAge report: This report uses the Factbook shared data source. It contains the Tablix data region to compare the average life expectancy based on the category where the Median age and GDP fall. The tablix data region consist of a row group **GDP of country** and a column group **Median Age** to display the data.

Top10CountriesByGdp report: This report uses the FactbookSortedByGdp shared data source and shows the Top 10 countries by GDP. It contains a List data region with an image in the Image control that uses the RangeBar function to create an ad-hoc horizontal bar chart.

Reels

CustomerMailingList report: This report uses the Reels shared data source. It includes a Container control along with TextBox and Barcode controls to display a mailing list of customers. The **Columns** property of the Body section is set to **3** to display mailing labels in three columns.

CustomerOrders report: This report uses the Reels shared data source. It contains the List and Table data regions that group data by **CustomerID** and **SaleID** respectively to display order information. The **SalesAmount** textbox of the Table has its **Value** property set to Sum function to calculate the total for each order (as a table group subtotal). The **YearTotal** textbox of the Table also has its **Value** property set to Sum function to calculate the total of pre-tax sales. The Reels logo that is displayed on the report is embedded within the Reels theme. The page number in the PageHeader section is reset every time a new customer is displayed. This report also contains a subreport, CustomerOrdersCoupon.rdlx.

DistrictReport: This report uses the Reels shared data source. It contains the Chart and Tablix data region to display the number of sold items and the profit for each month during the two year span (2004-2005) for the selected district. The Tablix data region consists of a row group SaleDate and a column group StoreName. The report parameter determines which district to display and uses the available values from the second report dataset **SalesData**.The **StoreName** textbox in the report body uses the drill-through link (the **Action** is set to **Jump to report**) that opens the **StoreReport.rdlx** with more information.

DistrictSales report: This report uses the Reels shared data source. It contains the BandedList data region that groups data by **SaleYear**, **DistrictID** and **RegionID** to display district sales details. **Sum** function is used to get the total sales at the District, Region, Year levels and also to display a grand total. The Reels logo that is displayed on the report is embedded within the Reels theme.

Filmography report: This report uses the Reels shared data source. It contains nested List data regions that group data by **MovieID** and **MoviePersonID** to return a distinct number of movies with the selected actors. The report contains two cascading parameters that narrow down the number of actors displayed in the list first based on the alphabet with which the actor's name begins and then based on the actor's name. The Reels logo that is displayed on the report is embedded within the Reels theme.

GenreSales report: This report uses the Reels shared data source. It contains a Tablix data region to display the units sold for each genre, in each year and each quarter. The Tablix report data consist of a row group **GenreName** and a column group **SaleDate.Year** to display the data. The report also uses the plain column **Chart** to display the number of titles sold for each genre. This report uses the multi value parameter that allows you to select more than one genre for displaying sales data.

GenreStatistics report: This report uses the Reels shared data source. Median and Mode aggregate functions are used to show the middle values in a set of data as well as the most commonly occurring value. It also uses the **ReelsConfidential.rdlx-master** report to provide standard page headers and footers. The Reels logo that is displayed on the report is embedded within the Reels theme.

MonthlySalesReport: This report uses the Reels shared data source. It contains a Table data region that groups data by **DistrictID**. The Textbox controls in the Table have their **Value** property set to expressions to display the total of sales for each district and region as well as the totals of all districts within a region for a given month. It also uses the Plain Line Chart with the data grouped and sorted by **Day** for each **SaleDate** to display sales and profit for the selected month. This report uses query based parameters to select the month and region for displaying data. The Reels logo that is displayed on the report is embedded within the Reels theme.

MovieCatalog report: This report uses the Reels shared data source. It contains the Image, TextBox, TableofContents and List controls to display the list of movies in a catalog. This report uses the TableofContents control to display, an organized hierarchy of the report heading levels and labels along with their page numbers, in the body of a report. It also uses the TextBox and Image control to display the layout of the report. The Reels logo that is displayed on the report is embedded within the Reels theme.

MovieReport: This report uses the Reels shared data source. It contains four List data regions with groupings. The **MovieList** is grouped by **MovieID**, the **GenreList** is used to display the genre names and is grouped by **GenreID**, the **CrewList** is used to display the title and is grouped by **CrewTitleID**, and finally the **CastNameList** is used to display the cast and crew and is grouped by **MoviePersonID**. This report uses cascading parameters. The first parameter asks to select which letter the movie titles starts with, and then the second parameter asks to select a movie to display. The **CrewName** textbox in the report Body uses the drill-through link (the **Action** is set to **Jump to report**) that opens **Filmography.rdlx** with more information on the selected person. The parameters of this report are passed to the Filmography.rdlx report. The Reels logo that is displayed on the report is embedded within the Reels theme.

RegionPerformance report: This report uses NWind.mdb database. It contains the Table data region that uses the **Region** value to group the report data and the **SalesAmount** value to sort the report data. It also uses filtering to filter the report data by the **RegionID** value. Textbox controls in the Table have their **Value** properties set to Sum functions to display the total of sales amount for each region. The Reels logo that is displayed on the report is embedded within the Reels theme.

ReorderList report: This report uses Reels shared data source. It contains the Table data region without any grouping. The Table detail row has its **BackgroundColor** property set to the expression to create a light yellow bar report. The Reels logo that appears on the report is embedded within the Reels theme.

SalesByMediaType report: This report uses the Reels shared data source. It contains the List data region that is grouped by Image to display data. The list also includes the Table data region that groups its data by **MediaID**. The report also contains the Plain Column Chart to display sales and profit by media type and embedded images for each category. This report uses the **ReelsConfidential.rdlx-master** report to render the report page header and footer.

SalesByRegion report: This report uses the Reels shared data source. It contains the Tablix data region and subtotals to display the number of units sold and profit for each region by year and quarter. The Tablix report data consist of a row group **Region** and a column group **SaleDate.Year** to display the data. The report also uses the Plain Column Chart to display the annual profit for each region. Data in the Chart is also grouped by **Region** and **SaleDate.Year**.

SalesReceipt report: This report uses the Reels shared data source. It contains a Table data region and three Container controls nested in the List data region. The List is grouped by **SaleID** to produce the body of the receipt. The **salesTaxLabel** and **totalSalesTax** text boxes use the Sum function that totals the amount due in the list and adds tax to

the sum of a field for the grand total due. The Reels logo that is displayed on the report is embedded within the Reels theme.

SalesReport: This report uses the Reels shared data source. It contains a Table and Chart data regions to display totals of sales and profit for the selected date range. The Table also uses the Data Bar function to plot the profits. The Chart and Table data is grouped by **Month** and **Year**. The **Month** textbox uses the drill-through link to display **MonthlySalesReport.rdlx** with more information on the selected month. This report uses parameters that allow the null value to select the range of dates. The Reels logo that is displayed on the report is embedded within the Reels theme.

StorePerformance report: This report uses the Reels shared data source. It identifies stores with profits above or below expectations. It also has two Image controls that display the database images and use the IconSet function. This report uses the **ReelsConfidential.rdlx-master** report to render page headers and footers.

StoreReport: This report uses the Reels shared data source. It contains the Table, List and Chart data regions to display sales for each employee. The Table and Chart data is grouped by **EmployeeID**. The Table uses hierarchical grouping to show the relationship between employee and supervisor for each store. The **FirstName** textbox has its **Padding> Left** property set to the Level function that leaves a space 15 pixels wide to the left of the control. This is the drill-down report where the **Visibility > Hidden** property of the Table detail row is set to **=Fields!Supervisor.Value <> 0** and the **Visibility > ToggleItem** property is set to the **FirstName** data field. The expression in the **Visibility > Hidden** property calculates whether the supervisor field returns 0, so only the supervisor's name is displayed initially. By clicking the toggle image next to the supervisor's name, the rows with details about employees are displayed. The report uses a cascade of parameter values - **Region**, **District** and **StoreNumber**. Each parameter depends on the value of the previous parameter and each comes from a separate dataset. The Reels logo that is displayed on the report is embedded within the Reels theme.

TopPerformers report: This report uses the Reels shared data source. It contains two Table data regions to display the top and bottom performers based on the movies sales. Each Table data is grouped by **MoviePersonID** and **MovieID** (nested grouping). The TopN filter is applied to one table and the BottomN filter is applied to another. The number of items returned by each of the filters is specified in report parameters. This report also uses two integer parameters to alter the number of items displayed in each table. The parameters use default values, which are passed to textboxes of the Table Headers. This is the drill-down report where the second Table Group Header in each Table has its **Visibility > Hidden** property set to **True** and the **ToggleItem** property set to the **PerformerName** textbox. By clicking the toggle image next to the name, the rows containing details about performers are displayed.

Others

AnnualPortfolioChart report: This report uses the MostPopulatedCountriesEnergyUsageByYear shared data source. It illustrates composite charts by using a Chart data region that shows Annual Stock Performance. The Trading Volume and Trading Value are plotted on two Y axes represented by two different chart types, Column and Line.

Financial Reports - BalanceSheetReport: This report displays the company's assets and liabilities. The report contains embedded JSON data. Table, Textbox, and Image controls are used to display data. The total assets and liabilities are calculated using expressions.

Financial Reports - CashFlowReport: This report displays the company's cash flow. The report contains embedded JSON data. Table, Textbox, Line, Image, and Shape controls are used to display data. The VB code in the script transforms the numerical values in the accounting format.

Financial Reports - IncomeStatementReport: This report displays the income statement for the Year End (December 2017). The report contains embedded JSON data. Table, Textbox, and Image controls are used to display data. The net income and total revenue are calculated using expressions.

Financial Reports - IncomeStatementReport2: This report displays the company's sales and expenses. The report

contains embedded JSON data. Table, Textbox, Image, and Shape controls are used to display data. Total sales, Gross profit, and Net profit are calculated using expressions.

Flight On-time Performance Report: This report shows on-time performance of US airlines. The report uses nested data regions bound to different datasets from the following data sources:

- States.xml - XML file stores information about geographical data of regions and states US.
- FlightDetails.csv - CSV file stores flight information - flight date, arrival time, departure time, etc.
- Airlines.json - JSON file stores unique airlines names corresponding to an airlineID.

The controls that are included in this report are List, Container, Table, and Tablix, where Container, Table, and Tablix are nested inside List. These controls show data as described below:

- List displays geographical information - Region, State Name, and Number of Airports, from States.xml data file through Tablix data region and Textbox control inside Container control.
- Table displays flight information - Number of flights, on-time arrival and departure details, from FlightDetails.csv.

The mapping for 'DataSet Joins' is created by adding a filter in Table on Region fields. This lists the flight information in Table control that flew in a particular region displayed in the List.

Medical Reports - BloodTestReport: This report displays the patient's blood test results. The report contains embedded XML data. Table, Textbox, and Image controls are used to display data. Results and Reference Intervals are calculated using expressions.

Medical Reports - PatientDiseaseSummaryReport: This report displays a summary of the patient's diseases and allergies. The report contains embedded JSON data. Table, Textbox, and Image controls are used to display data.

Telecom Reports - TelephoneBillSample: This report displays the company's telephone bill. The report contains embedded XML data. Table, Checkbox, Image, and Textbox controls are used to display data. Expressions are used to calculate totals.

Section Report folder

BarCode report: This report displays all barcode types that are supported by ActiveReports.

- **PageHeader section**: This section contains the table columns to display the report header. The **Text** property of the two Textbox controls, define the name of the two table columns.
- **Detail section**: This section contains a list of all the **Barcode** types that are supported by ActiveReports. The barcodes are presented in a table that contains two columns, the one with the **Textbox** control displays the barcode name, and the other one with the Barcode control displays the barcode image.

Invoice1 report: This report calculates the total amount for each customer along with purchase details. The report also displays the average of the previous invoice amount, the current invoice amount and the consumption tax. This report uses Bill.mdb database to provide data.

- **PageHeader section**: This section contains the **Label** and **Textbox** controls. It uses the bound data fields to calculate the invoice information in the header. The values of the **Excise** and **BillTotal** Textbox controls are set in the **Script** tab and are calculated in the BeforePrint event.

> **Note**: This sample uses bound fields on the PageHeader section, assuming that the value of all records to be displayed on a page does not change. However, it is not recommend that you place bound fields in the PageHeader or PageFooter sections because these sections are displayed on a page once. For the same reason it is not recommended that you place bound fields on the GroupHeader and GroupFooter sections.

- **GroupHeader section**: This section (ghColumnCaption) contains the Label controls that are captions for the information displayed in the Detail section. This section also uses the CrossSectionBox and the Shape controls.
- **Detail section**: This section contains the bound TextBox controls that display each row of data associated with the current ghColumnCaption. The Shape control is used to alternate row colors in the Detail section. Go to the **Script** tab and see the shpDetailBack.BackColor property in the Detail_Format event.
- **PageFooter**: This section contains the Label controls that explain codes used in the Category column of the invoice details.

Invoice2 report: This report shows another invoice layout and uses **Seikyu2.mdb** database to provide data.

- **GroupHeader1 section**: This section contains the Label and Textbox controls. It uses the bound data fields to calculate the invoice information in the header. The values of the **Year, Month** and **Day** Textbox controls are set under the **Script** tab and are calculated in the ReportStart event. It also uses the **CrossSectionLine** and the **CrossSectionBox** controls that span the GroupHeader1 section to Detail section. The CrossSection Box ends in the GroupFooter1 section. These controls form vertical lines between columns of the invoice details and a rectangle around the details of the invoice at run time.
- **Detail section**: This section uses the data table **tb_Main** for the main report data and the data table **tb_Count** to retrieve the number of data items within a group. The Detail data in each group is retrieved beforehand to calculate the required number of empty rows. For the required empty row count, substitute data with "" in the FetchData event. The **Shape** control is used to alternate row colors in the Detail section. Go to the **Script** tab and see the shpDetailBack.BackColor property in the Detail_Format event.
- **GroupFooter1 section**: This section contains the Label and Textbox controls that display the totals of the invoice. The values of the **Tax** and **Pretax** Textbox controls are set under the **Script** tab and are calculated in the Format event.

LabelReport: This report displays the tack seal, which is commonly used in postal services. This multi-column report uses the customer barcode that is bound to data by using the **DataField** property. This report uses **Nwind.mdb** database to provide data.

- **Detail1 section**: This section contains the bound data fields to display the contact information and the Barcode control that is bound to data by using the **DataField** property. The **ColumnCount** property is set to 3, which allows the report to display the tack seal in 3 columns.

PaymentSlip Report: The report displays the salary payment slip where the EAN128FNC1 barcode is used for barcode bound to a data field. This report uses **Bill.mdb** data source connection to provide data.

- **Detail section**: This section contains bound data fields to display the payment information and the Barcode control that is bound to data by using the **DataField** property.

PurchaseOrder report: The report displays purchase slips created with bound data. It groups data for each purchase slip. The detail count is not fixed but the number of rows in the report layout is fixed. In this case, when the **RepeatToFill** property of the Detail section is set to True, the detail data is repeatedly displayed on the entire page. The report uses the **RepeatToFill** property and calculated fields to demonstrate how to create the report layout without any code. Calculated fields are used to calculate the total cost and the total selling price. This report uses **Shiire.mdb** data source.

- **groupHeader1 section**: This section contains the Label and Textbox controls. It uses the bound data fields to calculate the company information in the header.
- **Detail section**: This section contains Textbox controls. It uses bound data fields to calculate the product information in the body of the report. The **detail.BackColor** property is used to alternate row colors in the Detail section. Go to the **Script** tab and see the detailBack.BackColor property in the detail_Format event.
- **groupFooter1 section**: This section contains the Label and Textbox controls. It uses the bound data fields to calculate the total information in the footer of the report group.

Schedule report: The sample report displays the weekly schedule of employees in the gantt chart format. The report displays six day report schedules on a single page in the Viewer control. This report uses **Schedule.mdb** database to provide data.

- **GroupHeader1 section**: This section contains the Label and Textbox controls. It uses the bound data fields to calculate the employee information in the header.
- **Detail section**: The Gantt chart view is created by using the Label, Shape and Line control. You can display the horizontal bars like in the gantt chart by modifying the size of the Label control. To add horizontal bars, you can dynamically add twenty four (12 x 2) Label controls within the 9-20 hours time frame that will become the horizontal bars of gantt chart in the ReportStart event. Once they are added, set the **Visible** property to False to hide them.

You can adjust the width of horizontal bars by setting calculated values from the time width in the **Tag** property of the Label control within Detail_Format event. For the horizontal adjustment of bars, change the width of the Label control to match with the value in the **Tag** property.
UnderlayNext report: This report displays the bound data. It uses **Nwind.mdb** database to provide data.

- **PageHeader section**: This section contains the Label and Line controls to create the header for report data.
- **GroupHeader1 section**: This section contains a TextBox control bound to the Country field and the CrossSectionLine controls to create the table for the body data of the report. **UnderlayNext Property (on-line documentation)** of this section is set to True to align the beginning position of the report data in the Detail section with the GroupHeader1 section.
- **Detail section**: This section contains the bound Textbox controls to display report data. This section displays the name of the city, contact name and postal code for each customer according to their countries.

Win Viewer

This sample demonstrates using Win Viewer to load RPX, RDL, RDLX, JSON or RDF report formats or save reports to report document file (RDF) format.

Sample Location

Visual Basic.NET

```
..\Samples14\Desktop\WinViewer\VB.NET
```

C#

```
..\Samples14\Desktop\WinViewer\C#
```

Details

When you run this sample, a Viewer control containing File and Help menu appears on the top. You can load any of the RDF files from **RDFs** folder using File > Open. An RDF file is a static copy of a report saved to the native Report Document Format. This can be loaded into the Viewer control without running it or accessing data. You can use **Save as RDF** option to save other report formats to RDF format. For more information, see Save and Load RDF Report Files. You can also export to other file formats using File > Export option.

WPF Viewer

The WPF Viewer samples demonstrates the use of WPF Viewer and its options to view the rdlx and rpx reports.

Sample Location

Visual Basic.NET

```
..\Samples14\Desktop\WPFViewer\VB.NET
```

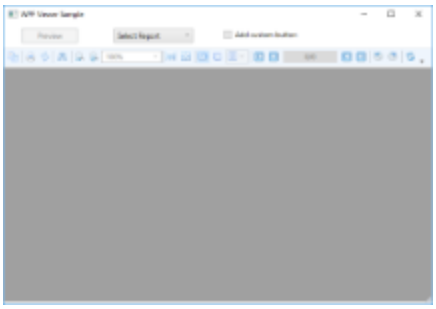
C#

```
..\Samples14\Desktop\WPFViewer\C#
```

Details

Run-Time Features

When you run the sample, MainWindow.xaml containing the WPF Viewer, **Select Report** ComboBox, **Preview** button and **Add Custom Button** CheckBox appears.



Select Report

In the **Select Report** ComboBox, you can select from a list of 6 sample reports. The ComboBox contains the following reports - Catalog.rdlx, EmployeeSales.rdlx, Invoice1.rdlx, Invoice2.rpx, LabelReport.rpx, and PaymentSlip.rpx.

Preview

The **Preview** button opens the report selected in **Select Report** ComboBox in the WPF Viewer.

Add custom button

The **Add custom button** CheckBox demonstrates the customization options of the WPF Viewer. To see the **About Us** custom button appear in the WPF Viewer toolbar, select the **Add custom button** CheckBox and click the **Preview** button. To remove the **About Us** custom button from the WPF Viewer toolbar, click to clear the **Add custom button** CheckBox and then click the **Preview** button.

> **Note: Preview** button and **Add custom button** CheckBox are only enabled when a report is selected from the **Select Report** ComboBox.

Project Details

Reports folder

The folder contains the following reports.

Catalog.rdlx: This is a sample layout for the product catalog. This report uses multiple page layouts to create a catalog. The layout of this report is spread over 4 pages, which appear one after another at run time. **Page1** and **Page2** layouts simply display introductive text. The layout on **Page3** contains a List data region with TextBox controls and a Table to display product details for each product category. The List is grouped by the CategoryID field to filter products by their category. The layout on **Page4** uses Textbox, Shape and Line controls to create a Order Form, which a user is to fill manually.

EmployeeSales.rdlx: This is a sample layout to display sales by each employee for the year 1997. The Chart control is used to display a graphical analysis of sales by each employee and the Table data region lists down the exact numbers.

Invoice1.rdlx: This report uses a BandedList and few TextBox controls to create the Invoice layout for displaying customer transactions. Both page and the BandedList control are grouped by the OrderID field. Texboxes in the footer section of the BandedList control use the Sum function to display the total of the transactions. For detailed information on the Invoice2.rpx report, see the Reports Gallery Sample.

Invoice2.rpx: This is a sample layout for the invoice report. The report page is grouped by the EstimateID field. Sum function is used to display the GrandTotal of all transactions. For detailed information on the Invoice2.rpx report, see the Reports Gallery Sample.

LabelReport.rpx: This report displays the tack seal, which is commonly used in postal services. This multi-column report uses the customer barcode that is bound to the data using the **DataField** property. For detailed information on the LabelReport.rpx report, see the Reports Gallery Sample.

PaymentSlip.rpx: This report displays the invoice payment slip with the GS1-128 barcode that is used for payment services in convenience stores. GS1-128 is the convenience store barcode, formerly called UCC/EAN-128. For detailed information on the PaymentSlip.rpx report, see the Reports Gallery Sample.

DefaultWPFViewerTemplates.Xaml

The DefaultWPFViewerTemplates.xaml is used for the WPF Viewer customization. For steps on the WPF Viewer customization, see the WPF Viewer walkthrough.

MainWindow.xaml

The MainWindow.xaml is displayed when you run the sample. It contains the WPF Viewer, the **Select Report** ComboBox, the **Preview** button and the **Add custom button** CheckBox.

The code behind of MainWindow.xaml.vb (or .cs), handles the display of RDLX and RPX reports in the WPF Viewer and the customization of the WPF Viewer application.

MyCommand

This class contains the text that is displayed when you click the **About Us** custom button in the WPF Viewer toolbar.

Web

The Web folder contains following sample:

Custom Preview
: The sample demonstrates exporting an ActiveReports report to the HTML or PDF format in your Web application.

Custom Preview

The Custom Preview sample demonstrates a method to view a report at client side in HTML or PDF format. Application structure consists of ASP.NET website, using which the report is streamed to the client as HTML or PDF. This sample describes custom exporting without the Pro Edition server controls or RPX handlers as well as running reports on the server. The PDF and HTML exports allow you to manually control exporting by writing a little code in ASP.NET language. Steps explained in this sample can be used for both Standard and Professional editions.

Note: Before running this sample, in the Solution Explorer, click the Licenses.licx file and then, from the **Build** menu, select Build Runtime License. Please see To license Web Forms projects made on the trial version for details.

Sample Location

Visual Basic.NET

```
..\Samples14\Web\CustomPreview\VB.NET
```

C#

```
..\Samples14\Web\CustomPreview\C#
```

Details

When you run the sample, the Default.aspx page appears in your browser. This page provides two links to other reports that demonstrate custom PDF or HTML export options.

Clicking the **Custom Exporting PDF Example** option opens the **Invoice** report and clicking **Custom Exporting HTML Example** option opens **NwindLabels** report in the Default.aspx page.

> **Note:** To run this sample, you must have Nwind.mdb downloaded from GitHub in *..\Samples14\Data\NWIND.mdb*

The project consists of the following elements.

- Reports folder: The Reports folder contains two rpx reports - the **Invoice** report and the **NwindLabels** report.

  Invoice Report

  The Invoice report uses three GroupHeader sections, a Detail section and a GroupFooter section as well as a label in the PageFooter section to display data.

  > **Note:** Except for the Detail section, all sections come in header and footer pairs. Unused sections have their **Height** properties set to **0** and their **Visible** properties set to **False**.

  ghOrderHeader section

  The **DataField** property of this section is set to **OrderID**. This setting, in conjunction with data ordered by the OrderID field, causes the report to print all of the information for one order ID value, including all of the related details and footers, before moving on to the next order ID. For more information on grouping, see Add Groups.

  This section also contains a Picture control, a number of Label controls, and two bound TextBox controls. The TextBoxes are bound using the **DataField** property in the Properties window, and the date is formatted using the **OutputFormat** property.

  ghOrderID section

  The **DataField** property of this section is also set to **OrderID**. This allows subtotal summary functions in the related GFOrderID section to calculate properly.

  This section contains a number of labels and bound text boxes, as well as two **Line** controls.

  ghTableHeader section

  This section contains only labels for the data to follow in the Detail section.

  Detail section

  This section contains bound TextBox controls. These TextBoxes render once for each row of data found in the current OrderID before the report moves on to the GroupFooter sections.

  GFOrderID section

The **NewPage** property of this section is set to **After**. This causes the report to break to a new page and generate a new invoice after this section prints its subtotals.

This section contains several labels and several TextBoxes. Two of the TextBox controls use the following properties to summarize the detail data: **SummaryFunc**, **SummaryGroup**, and **SummaryType**. For more information, Create a Summary Report.

The **Total** TextBox does not use the DataField property or any of the summary properties, or even any code. To find the functionality of this TextBox, in design view, click the **Script** tab at the bottom of the report.

PageFooter section

This section has one simple Label control. For more information about report sections and the order in which they print, see Section Report Structure and Section Report Events.

NwindLabels Report

TheNwindLabels report only uses the Detail section to display the report data.

> **Note:** Except for the Detail section, all sections come in header and footer pairs. Unused sections have their **Height** properties set to **0** and their **Visible** properties set to **False**.

Detail section

This section contains bound TextBox controls and Label controls. This section prints 30 labels per 8½ x 11 sheet.

The Detail section uses the **CanGrow** property set to **False** to maintain the label size and the **ColumnCount**, **ColumnDirection,** and **ColumnSpacing** properties to accommodate multiple labels in a single page.

- CustomExportHtml.aspx: This Web form is displayed by clicking the **Custom Exporting HTML Example** option on the Default.aspx page. In CustomExportHtml.aspx, report is outputted to the ReportOutput folder using the **CustomHtmlOutput** class and the exported HTML is displayed in the browser. The CustomHtmlOutput class implements the required IOutputHtml in the HTML export and saves the output results to a file with a unique name.

  > **Note:** This sample requires write permissions to the **ReportOutput** folder that is located in the web samples directory.

- CustomExportPdf.aspx: The Web form is displayed by clicking the **Custom Exporting PDF Example** option on the Default.aspx page. In CustomExportPdf.aspx, the report is exported to memory stream and then outputted in the browser.

  > **Note:** This sample requires write permissions to the **ReportOutput** folder that is located in the web samples directory.

- Default.aspx: This is the main Web form of the sample that shows the introductory text and links to other sample pages that demonstrate the following web features.
  - **Custom Exporting PDF Example** - This link opens the Invoice report in the PDF Reader by exporting it to memory stream and then outputting it in the browser.
  - **Custom Exporting HTML Example** - This link opens the NWindLabels report. This report is outputted to the ReportOutput folder by using the CustomHtmlOutput class and the exported HTML is displayed in the browser.

- Web.config: This configuration file contains the httpHandlers that allow ActiveReports to process reports on the Web. Note that you need to manually update version information here when you update your version of

ActiveReports.

Web Samples

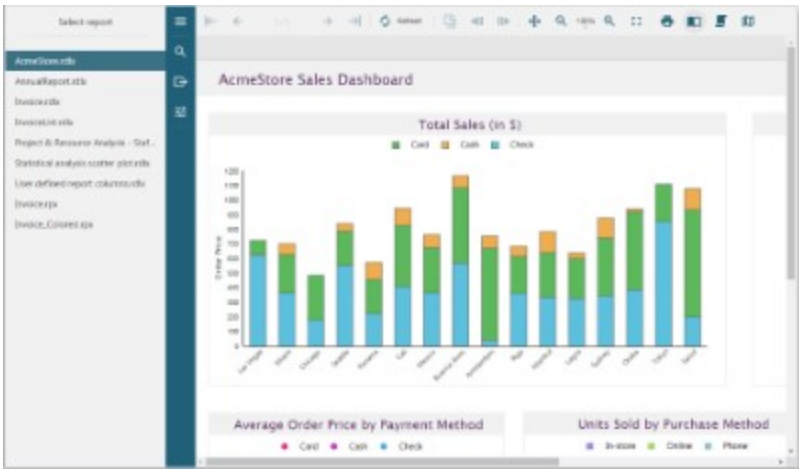
The samples in WebSamples folder demonstrate web related features in JSViewer, Web Designer, etc. Download these samples from following link:
https://github.com/activereports/WebSamples14

Sample	Description
JSViewer Angular(Core)	This sample demonstrates the use of the GrapeCity ActiveReports JSViewer with an Angular 8 app and ASP.NET Core back end.
JSViewer MVC	This sample demonstrates the use of GrapeCity ActiveReports JSViewer with an ASP.NET MVC 5 back end.
JSViewer MVC(Core)	This sample demonstrates the use of GrapeCity ActiveReports JSViewer with an ASP.NET MVC Core back end.
JSViewer React(Core)	This sample demonstrates the use of GrapeCity ActiveReports JSViewer with an ReactJS app and ASP.NET Core back end.
JSViewer Vue(Core)	This sample demonstrates the use of GrapeCity ActiveReports JSViewer with an VueJS app and ASP.NET Core back end.
JSViewer Blazor	The sample demonstrates the use of the GrapeCity ActiveReports JSViewer and the Blazor Framework that allows building client-side web applications with C#.
Web Designer MVC	This sample demonstrates Web Designer with an ASP.NET MVC 5 back end.
Web Designer MVC(Core)	This sample demonstrates Web Designer with an ASP.NET MVC Core back end.
Web Designer Angular(Core)	This sample demonstrates the use of GrapeCity ActiveReports Web Designer with an Angular 8 app and ASP.NET Core backend.
WebViewer Pro	This Active Reports Web Pro Sample demonstrates the use of Professional Edition ASP.NET features, such as HTTP handlers, parameterized reports, and more.

Important: It is recommended to use **Visual Studio 2019** for applications running on ASP.NET Core. This is because Visual Studio 2017 does not support core license compilation, and you will see 'No License' message on running ASP.NET Core applications on Visual Studio 2017.

JSViewer Angular(Core)

The JSViewer_Angular(Core) sample demonstrates the use of the GrapeCity ActiveReports JSViewer with an Angular 8 app and ASP.NET Core back-end.



Note: To run this sample, you must have

- Visual Studio 2019 (https://visualstudio.microsoft.com/vs/) version 16.4 or newer.
- .NET Core 3.1 SDK (https://www.microsoft.com/net/download) or later installed on your machine.
- .NET Core Hosting Bundle (https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/iis/?view=aspnetcore-3.1#install-the-net-core-hosting-bundle) (for deployment to IIS).
- Angular 8 requires Node.js (https://nodejs.org) 10 or later.

Sample Location

```
..\WebSamples14\JsViewerSamples\JSViewer_Angular(Core)
```

Details

When you run the sample, the default page appears in your browser. This page provides links to reports that demonstrate the use of the GrapeCity ActiveReports JSViewer with an Angular 7 app and ASP.NET Core back-end.

Clicking the report link in the left panel opens the report for preview. You can preview the following reports.

- AnnualReport.rdlx
- Invoice.rdlx
- Invoice.rpx
- InvoiceList.rdlx
- Invoice_Colored.rpx
- Project&ResourceAnalysis
- Statistical analysis scatter plot.rdlx - Staff Performance Analysis.rdlx
- User defined report columns.rdlx

Note: The timeout error sometimes appears on running the JSViewer_Angular(Core) sample with default settings. In this case, you should increase the connection timeout period. See Troubleshooting for details on how to resolve this issue.

The project consists of the following elements.

- ClientApp folder: This folder contains a standard Angular CLI app that is used for all UI concerns.
- Controllers folder: This folder contains the **ReportsController** files. The **ReportsController** handles the interaction with reports when a report is selected in the left panel.

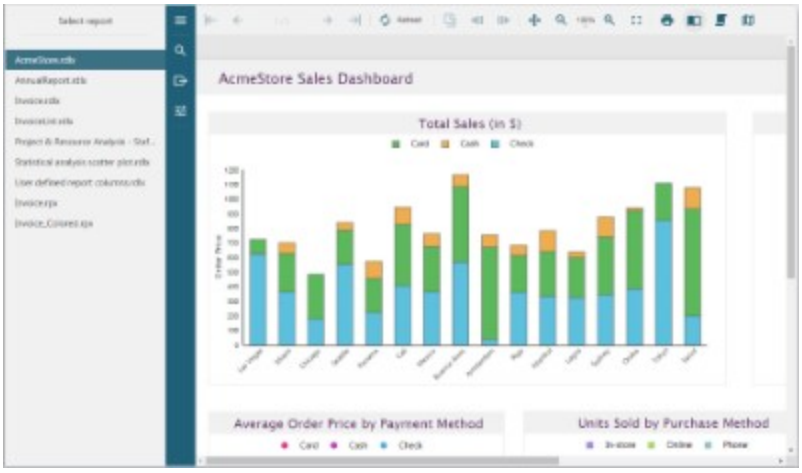
- appsettings.json: The json configuration file.
- readme: This file contains the instructions on how to run the sample project.
- Startup.cs: This is the default startup file.
- Web.config: This configuration file contains the httpHandlers that allow ActiveReports to process reports on the Web. Note that you need to manually update version information here when you update your version of ActiveReports.

⚠ Before publishing the sample, you must do the following.

- In the **JSViewer_Angular(Core).csproj** file, set the **PublishToIIS** property to true as follows: `<PublishToIIS>true</PublishToIIS>`
- Copy the sample **ViewerApp** folder to the **publish** folder.

JSViewer MVC

The JSViewer_MVC sample demonstrates the use of the GrapeCity ActiveReports JSViewer with an ASP.NET MVC 5 back-end.



Note: To run this sample, you must have:

- Visual Studio 2013 (https://visualstudio.microsoft.com/vs/) or newer
- .NET Framework Dev Pack (https://dotnet.microsoft.com/download) 4.6.2 or later

Sample Location

`..\WebSamples14\JsViewerSamples\JSViewer_MVC`

Details

When you run the sample, the default page appears in your browser. This page provides links to reports that demonstrate the use of the GrapeCity ActiveReports JSViewer with an ASP.NET MVC 5 backend.

Clicking the report link in the left panel opens the report for preview. You can preview the following reports.

- AnnualReport.rdlx
- Invoice.rdlx

- Invoice.rpx
- Invoice_Colored.rpx
- InvoiceList.rdlx
- Project & Resource Analysis - Staff Performance Analysis.rdlx
- User defined report columns.rdlx
- Statistical analysis scatter plot.rdlx

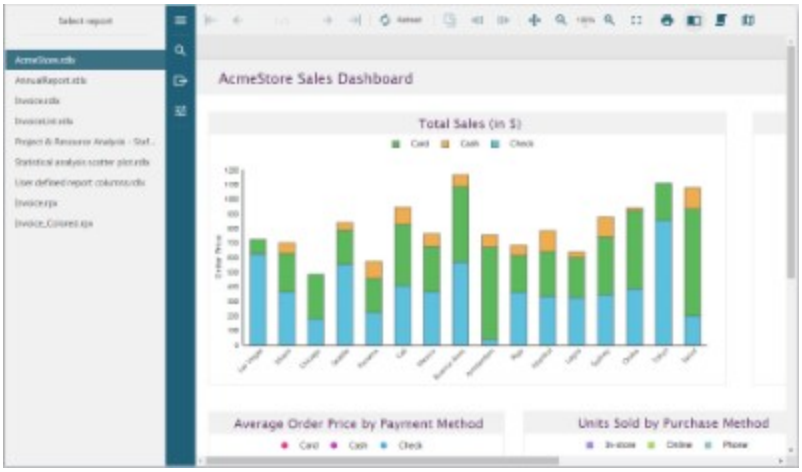
The project consists of the following elements.

- Controllers folder: This folder contains the **HomeController** that handles the user interaction and returns the main Index view.
- ViewerApp folder: Contains JSViewer CSS and JavaScript files.
- Global.asax: The default class that sets global URL routing values for this web application.
- packages.config
- Startup.cs: This is the default startup file.
- Web.config: This configuration file contains the httpHandlers that allow ActiveReports to process reports on the Web. Note that you need to manually update version information here when you update your version of ActiveReports.

⚠ Before publishing the sample, you must copy the sample **ViewerApp** folder to the **publish** folder.

JSViewer MVC(Core)

The JSViewer_MVC(Core) sample demonstrates the use of the GrapeCity ActiveReports JSViewer with an ASP.NET Core back-end.



Note: To run this sample, you must have

- Visual Studio 2019 (https://visualstudio.microsoft.com/vs/) version 16.4 or newer
- .NET Core 3.1 SDK (https://www.microsoft.com/net/download) or later installed on your machine.
- .NET Core Hosting Bundle](https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/iis/?view=aspnetcore-3.1#install-the-net-core-hosting-bundle) (for deployment to IIS)

Sample Location

`..\WebSamples14\JsViewerSamples\JSViewer_MVC(Core)`

Details

When you run the sample, the JSViewer opens in your browser. The viewer provides links to reports to demonstrate GrapeCity ActiveReports JSViewer with an ASP.NET Core back-end.

Clicking the report link in the left panel opens the report for preview. You can preview the following reports.

- AnnualReport.rdlx
- Invoice.rdlx
- Invoice.rpx
- InvoiceList.rdlx
- Invoice_Colored.rpx
- Project & ResourceAnalysis - Staff Performance Analysis.rdlx
- Statistical analysis scatter plot.rdlx
- User defined report columns.rdlx

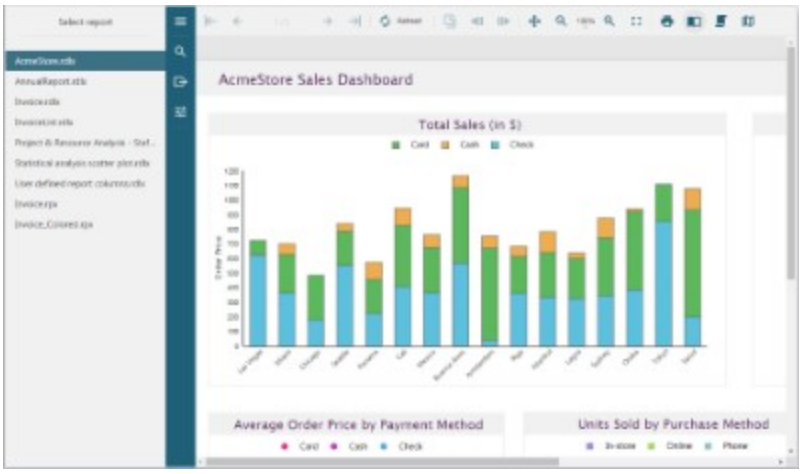
The project consists of the following elements.

- Controllers folder: This folder contains the **HomeController** that handles the user interaction and returns the main Index view.
- ViewerApp folder: Contains JSViewer CSS and JavaScript files.
- readme: This file contains the instructions on how to build and run the sample project.
- Startup.cs: This is the default startup file.
- Web.config: This configuration file contains the httpHandlers that allow ActiveReports to process reports on the Web. Note that you need to manually update version information here when you update your version of ActiveReports.

> ⚠ Before publishing the sample, you must do the following.
>
> - In the **index.html** file, uncomment the following line:
>   `<!--<base href="/JSViewer_MVC_Core/">-->`
> - Copy the sample **ViewerApp** folder to the **publish** folder.

JSViewer React(Core)

This sample demonstrates the use of GrapeCity ActiveReports JSViewer with a ReactJS app and the ASP.NET Core back end.



Note: To run this sample, you must have

- Visual Studio 2019 (https://visualstudio.microsoft.com/vs/) version 16.4 or newer
- .NET Core 3.1 SDK (https://www.microsoft.com/net/download) or later installed on your machine.
- .NET Core Hosting Bundle](https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/iis/?view=aspnetcore-3.1#install-the-net-core-hosting-bundle) (for deployment to IIS)
- Node.js 8.x or 10.x.

Sample Location

```
..\WebSamples14\JsViewerSamples\JSViewer_React(Core)
```

Details

When you run the sample, the JSViewer opens in your browser. The viewer provides links to reports to demonstrate GrapeCity ActiveReports JSViewer with a ReactJS app and the ASP.NET Core back end.

Clicking the report link in the left panel opens the report for preview. You can preview the following reports.

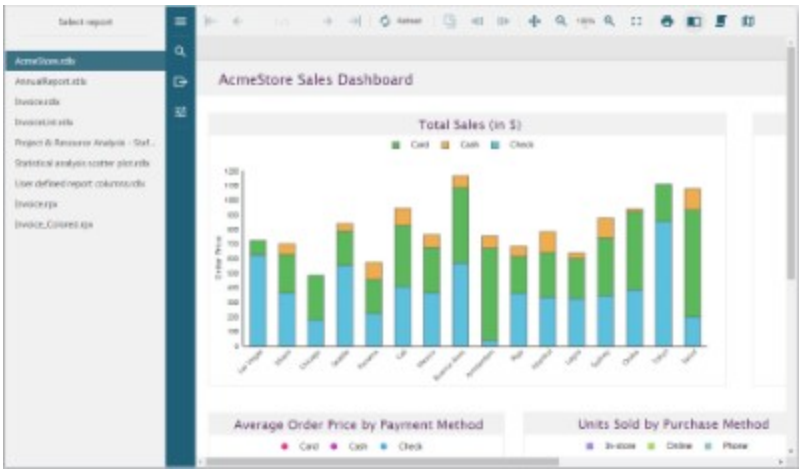
- AcmeStore.rdlx
- AnnualReport.rdlx
- Invoice.rdlx
- Invoice.rpx
- InvoiceList.rdlx
- Invoice_Colored.rpx
- Project & ResourceAnalysis - Staff Performance Analysis.rdlx
- Statistical analysis scatter plot.rdlx
- User defined report columns.rdlx

The project consists of the following elements.

- **Controllers** folder: This folder contains the **HomeController** that handles the user interaction and returns the main Index view.
- **ClientApp** folder: This folder contains a standard Angular CLI app that is used for all UI concerns.
- **readme**: This file contains the instructions on how to build and run the sample project.
- **Startup.cs**: This is the default startup file.
- **wwwroot**: Contains designer CSS and JavaScript files.

JSViewer Vue(Core)

The sample demonstrates the use of GrapeCity ActiveReports JSViewer with an VueJS app and ASP.NET Core back end.



Note: To run this sample, you must have

- Visual Studio 2019 version 16.4 or newer.
- .NET Core 3.1 SDK (https://dotnet.microsoft.com/download) or later installed on your machine.
- .NET Core Hosting Bundle (https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/iis/?view=aspnetcore-3.1#install-the-net-core-hosting-bundle) (for deployment to IIS).
- Node.js 8.x or 10.x.

Sample Location

```
..\WebSamples14\JsViewerSamples\JSViewer_Vue(Core)
```

Details

When you run the sample, the JSViewer opens in your browser. The viewer provides links to reports to demonstrate GrapeCity ActiveReports JSViewer with a ReactJS app and the ASP.NET Core back end.

Clicking the report link in the left panel opens the report for preview. You can preview the following reports.

- AcmeStore.rdlx
- AnnualReport.rdlx
- Invoice.rdlx
- Invoice.rpx
- InvoiceList.rdlx
- Invoice_Colored.rpx
- Project & ResourceAnalysis - Staff Performance Analysis.rdlx
- Statistical analysis scatter plot.rdlx
- User defined report columns.rdlx

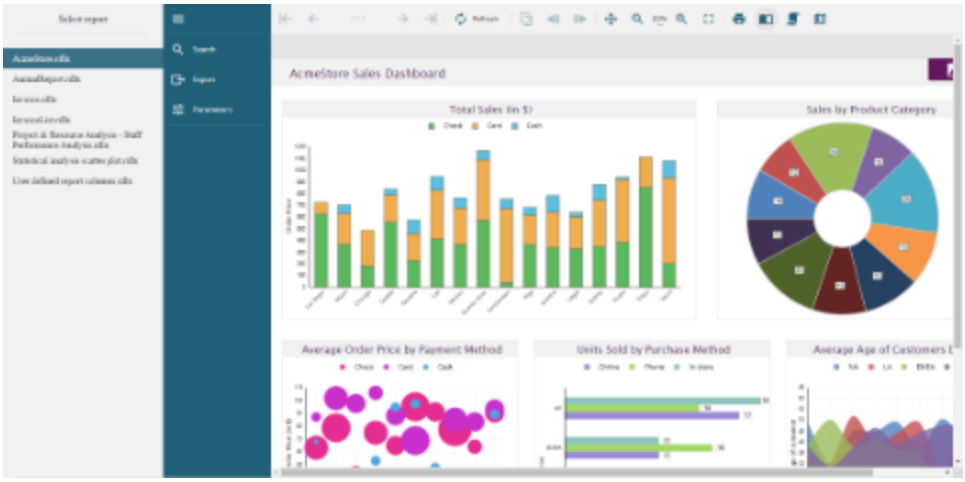
The project consists of the following elements.

- **Controllers** folder: This folder contains the **HomeController** that handles the user interaction and returns the main Index view.

- **ClientApp** folder: This folder contains a standard Angular CLI app that is used for all UI concerns.
- **readme**: This file contains the instructions on how to build and run the sample project.
- **Startup.cs**: This is the default startup file.
- **wwwroot**: Contains designer CSS and JavaScript files.

JSViewer Blazor

The sample demonstrates the use of the GrapeCity ActiveReports JSViewer and the Blazor Framework that allows building client-side web applications with C#.



Note: To run this sample, you must have

- Visual Studio 2019 (https://visualstudio.microsoft.com/vs/) version 16.4 or newer.
- .NET Core 3.1 SDK (https://dotnet.microsoft.com/download) or later installed on your machine.
- .NET Core Hosting Bundle (https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/iis/?view=aspnetcore-3.1#install-the-net-core-hosting-bundle) (for deployment to IIS).

Sample Location

..\WebSamples14\JsViewerSamples\JSViewer_Blazor

Details

When you run the sample, the JSViewer opens in your browser and provides links to reports to demonstrate GrapeCity ActiveReports JSViewer in a Blazor application.

Clicking the report link in the left panel opens the report for preview. You can preview the following reports.

- AcmeStore.rdlx
- AnnualReport.rdlx
- Invoice.rdlx
- InvoiceList.rdlx
- Project & ResourceAnalysis - Staff Performance Analysis.rdlx
- Statistical analysis scatter plot.rdlx

- User defined report columns.rdlx

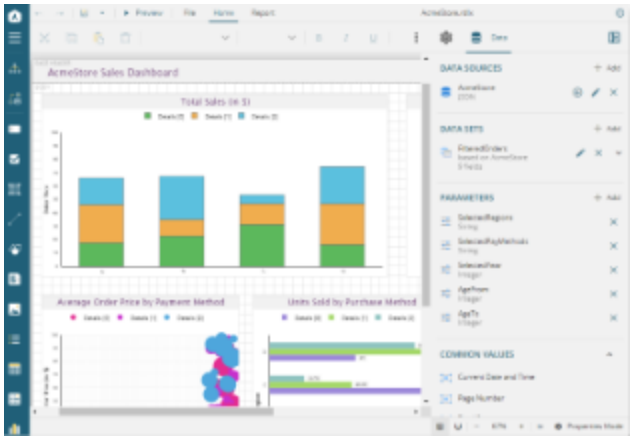
The project uses the **GrapeCity.ActiveReports.Aspnetcore.Viewer** Nuget package. See Manage ActiveReports Dependencies for details.

The project consists of the following elements.

- **wwwroot**: Contains static CSS and JavaScript files of the Blazor application, which are copied from the ARJS Viewer npm package (https://www.npmjs.com/package/@grapecity/ar-viewer).
- **Data** folder: This folder contains the ReportsService.cs.
- **Pages** folder: This folder contains Razor pages and supporting files.
- **Reports** folder: This folder contains all available reports.
- **_Imports.razor**: The Razor template file.
- **appsettings.json**: The json configuration file.
- **Startup.cs**: This is the default startup file. It contains the **ConfigureServices** method, which configures the application so that it can handle cross-domain requests.

Web Designer MVC

The Web Designer_MVC sample demonstrates Web Designer with an ASP.NET MVC 5 back-end.



Sample Location

```
..\WebSamples14\WebDesignerSamples\Web Designer_MVC
```

Details

When you run the sample, the Web Designer opens in your browser wherein you can create, edit, or modify your reports. Following are the main menu options:

- **File**: Contains options to create, open, or save reports. It also contains the version information in About option and help documentation link in Help option.
- **Home**: Consists of report editing options such as cut, copy, paste, and delete. It also provides shortcuts for text formatting such as font, font size, font color, and horizontal and vertical text alignments.
- **Report**: Contains options to add, delete, or move pages (in Page report) and add or remove header and footer

(RDL report), and change report themes.

- **Preview**: Click Preview to preview reports.
- **Properties**: Displays the properties of the selected report element. If more than one element is selected, only their common properties are shown.
- **Data**: Contains options to manage daya sources, data sets, and parameters. It also displays common values such as current date and time, page number, total pages, and more.

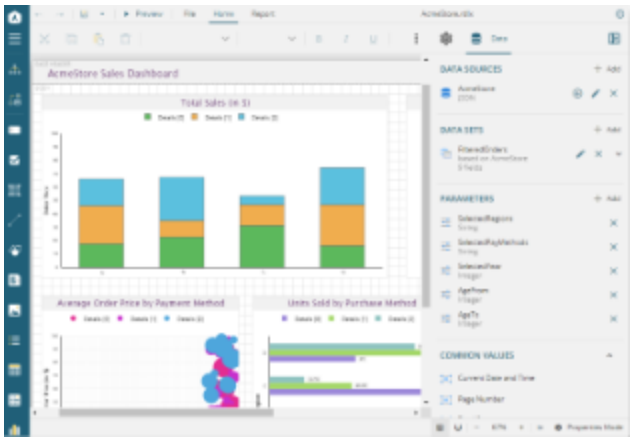
The project consists of the following elements.

- Controllers folder: Contains DataSets, Design, Preview, and Templates controllers.
- Datasets folder: Contains JSON datasets Categories, Employees, and Products.
- Resources folder: Contains reports, themes, images, etc. that are used by the project to illustrate Web Designer.
- Services folder: Consists of classes that get datasets and templates information.
- wwwroot folder: Contains designer CSS and JavaScript files.

> ⚠ Before publishing the sample, you must copy the sample **datasets**, **resources**, **templates** folders to the **publish** folder.

Web Designer MVC(Core)

The Web Designer_MVC(Core) sample demonstrates Web Designer with an ASP.NET Core back-end.



Sample Location

```
..\WebSamples14\WebDesignerSamples\Web Designer_MVC(Core)
```

Details

When you run the sample, the Web Designer opens in your browser wherein you can create, edit, or modify your reports. Following are the main menu options:

- **File**: Contains options to create, open, or save reports. It also contains the version information in About option and help documentation link in Help option.
- **Home**: Consists of report editing options such as cut, copy, paste, and delete. It also provides shortcuts for text

formatting such as font, font size, font color, and horizontal and vertical text alignments.

- **Report**: Contains options to add, delete, or move pages (in Page report) and add or remove header and footer (RDL report), and change report themes.
- **Preview**: Click Preview to preview reports.
- **Properties**: Displays the properties of the selected report element. If more than one element is selected, only their common properties are shown.
- **Data**: Contains options to manage daya sources, data sets, and parameters. It also displays common values such as current date and time, page number, total pages, and more.

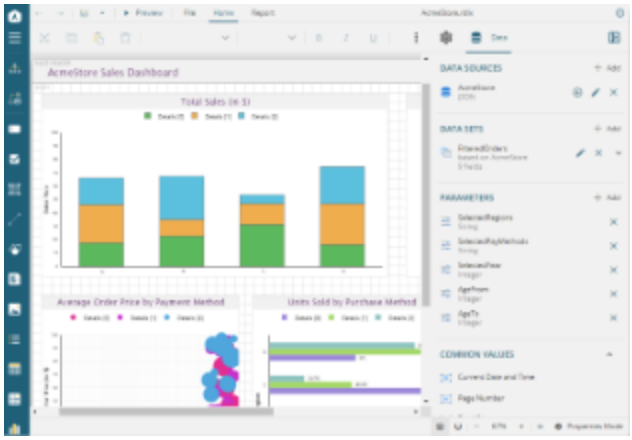
The project consists of the following elements.

- Controllers folder: Contains DataSets, Design, Preview, and Templates controllers.
- Datasets folder: Contains JSON datasets - Categories, Employees, Products, and DataSet with Parameters.
- Resources folder: Contains reports, themes, images, etc. that are used by the project to illustrate Web Designer.
- Services folder: Consists of classes that get datasets and templates information.
- wwwroot folder: Contains designer CSS and JavaScript files.

> ⚠ Before publishing the sample, you must copy the sample **datasets**, **resources**, **templates** folders to the **publish** folder.

Web Designer Angular(Core)

The WebDesigner_Angular(Core) sample demonstrates the use of GrapeCity ActiveReports Web Designer with an Angular 8 app and ASP.NET Core backend.



> **Note**: To run this sample, you must have
>
> - Visual Studio 2019 (https://visualstudio.microsoft.com/vs/) version 16.4.
> - .NET Core 3.1 SDK (https://dotnet.microsoft.com/download) or later installed on your machine.
> - .NET Core Hosting Bundle (https://dotnet.microsoft.com/download/dotnet-core/thank-you/runtime-aspnetcore-3.1.2-windows-hosting-bundle-installer) (for deployment to IIS)
> - Angular 8 requires Node.js (https://nodejs.org) 10 or later.

Sample Location

```
..\WebSamples14\JsViewerSamples\WebDesigner_Angular(Core)
```

Details

When you run the sample, the Web Designer opens in your browser wherein you can create, edit, or modify your reports. Following are the main menu options:

- **File**: Contains options to create, open, or save reports. It also contains the version information in About option.
- **Home**: Consists of report editing options such as cut, copy, paste, and delete. It also provides shortcuts for text formatting such as font, font size, font color, and horizontal and vertical text alignments.
- **Report**: Contains options to add, delete, or move pages (in Page report) and add or remove header and footer (RDL report), and change report themes.
- **Preview**: Click Preview to preview reports.
- **Properties**: Displays the properties of the selected report element. If more than one element is selected, only their common properties are shown.
- **Data**: Contains options to manage daya sources, data sets, and parameters. It also displays common values such as current date and time, page number, total pages, and more.

> **Note**: The timeout error sometimes appears on running the WebDesigner_Angular(Core) sample with default settings. In this case, you should increase the connection timeout period. See Troubleshooting for details on how to resolve this issue.

The project consists of the following elements.

- ClientApp folder: This folder contains a standard Angular CLI app that is used for all UI concerns.
- Controllers folder: Contains DataSets and Templates controllers.
- Datasets: Contains JSON datasets - Categories, Employees, Products, and DataSet with Parameters.
- Resources: Contains reports, themes, images, etc that are used by the project to illustrate Web Designer.
- appsettings.json: The json configuration file.
- readme: This file contains the instructions on how to run the sample project.
- Startup.cs: This is the default startup file.
- Web.config: This configuration file contains the httpHandlers that allow ActiveReports to process reports on the Web. Note that you need to manually update version information here when you update your version of ActiveReports.
- Services: Consists of classes that get datasets and templates information.
- wwwroot: Contains designer CSS and JavaScript files.

WebViewer Pro

The Active Reports WebViewer Pro Sample describes the standard ActiveReports web features as well as other features available in the Professional Edition only, such as HTTP handlers, custom PDF and HTML export, parameterized reports, and more.

Note: Before running this sample, in the Solution Explorer, click the Licenses.licx file and then, from the Build menu, select Build Runtime License. Please see To license Web Forms projects made on the trial version for details.

Sample Location

Visual Basic.NET

```
..\WebSamples14\JsViewerSamples\WebViewerPro_ASP.NET_VB.NET
```

C#

```
..\WebSamples14\JsViewerSamples\WebViewerPro_ASP.NET_C#
```

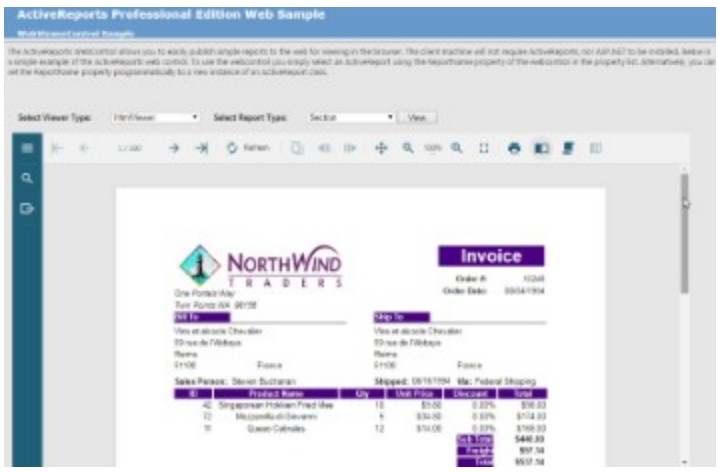
Details

When you run the sample, the Default.aspx page appears in your browser. This page provides links to other sample pages that demonstrate the following web features.

WebControl for ASP.NET: This link opens the WebControl.aspx page that allows you to select any of the three Viewer Types and it also allows you to select from Section, Page and RDL Report Type.

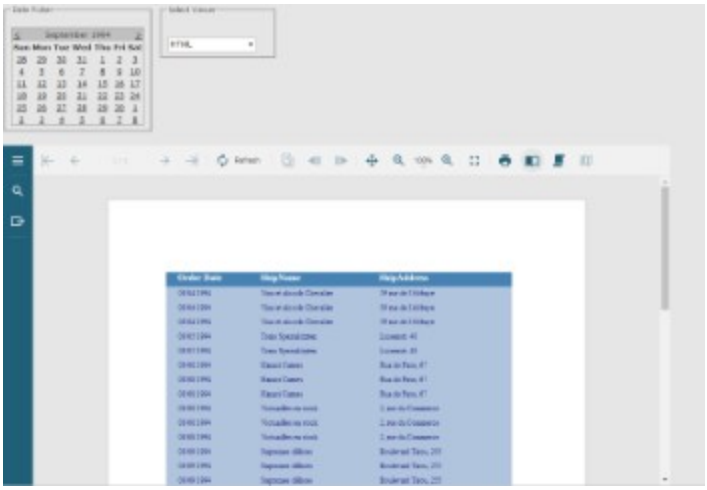
The three Viewer Types that are available are as follows:

- HtmlViewer
- AcrobatReader
- RawHtml



HTTPHandlers: This link opens the HttpHandlers.aspx page with the http handler examples.

Parameterized Report Example: This link opens the page that demonstrates how to generate a report by passing a parameter to the report.



> **Note:** To run this sample, you must have access to the **Nwind.mdb**. The NWIND.mdb file can be downloaded from GitHub: ..\Samples14\Data\NWIND.mdb.

The project contains the following elements:

- CodeReports: The CodeReports folder contains the following reports - **Invoice**, **InvoiceFiltered**, **NwindLabels** and **NwindLabelsFiltered**. The code reports are used to demonstrate how the ActiveReports Compiled Report HttpHandler functions. For more details, see **HttpHandlers.aspx** below.
- PageReports: The PageReprorts folder contains the **Invoice_Grouped** and **PurchaseReport** report.
- RDLReports: The RdlxReports folder contains the **SalesReceipt** report.
- RPXReports: The RpxReports folder contains the following reports - Invoice, **InvoiceFiltered**, **NwindLabels**, **NwindLabelsFiltered** and **Params**. These reports are used to demonstrate how the RPX HttpHandlers function. For more details, see **HttpHandlers.aspx** below.

  The Invoice.rpx report is used to demonstrate the WebViewer control options and is opened by clicking WebControl for ASP.NET on the Default.aspx page. This report is also opened by clicking the Custom Exporting PDF Example option on the Default.aspx page. For detailed information on the Invoice report, see the Cross Section Control Sample.

  The NwindLabels report is opened by clicking the Custom Exporting HTML Example option on the Default.aspx

page.

The Params report is used by the ParameterReport.aspx page to demonstrate how to generate a report by passing a parameter to the report.

- Default.aspx: This is the main Web form of the sample that shows the introductory text and links to the following sample pages.
  - WebControl for ASP.NET (WebControl.aspx)
  - HTTPHandlers (HttpHandlers.aspx)
  - Custom Exporting PDF Example (Invoice report)
  - Custom Exporting HTML Example (NWindLabels report)
  - Parameterized Report Example(ParameterReport.aspx)
- HttpHandlers.aspx: ActiveReports provides HttpHandler components that allow ASP.NET to automatically process reports that have been placed into an ASP.NET web site folder. ActiveReports HttpHandler components enable easily deployable reports in both HTML and PDF file formats. ActiveReports includes a simple configuration utility to properly register the HttpHandler components with IIS and ASP.NET.
  The RPX and RDLX HttpHandler processes and outputs reports from ActiveReports layout files (ending in the .rpx/.rdlx extension). When the ASP.NET receives a request for an ActiveReport file ending with the .rpx/.rdlx extension, the RPX/RDLX HttpHandler will run, and return the report's output in a format of your choice.
  The compiled Report HttpHandler enables easy distribution of ActiveReports that use compiled .NET source code. Compiled reports are exposed as a .NET class in a .NET assembly file. When ASP.NET receives a request for a file with the .ActiveReport extension, the Compiled Report handler will load the ActiveReports from the assembly, run it, and return the output in a format of your choice.
  For information on configuring the http handlers, see Configure HttpHandlers in IIS 8 and IIS 10. The required mapping for each feature has been listed below.

WebViewer control	ActiveReports 14 Cache Item Script Mapping is required
Compiled Report Handler (the report explorer is embedded in the assembly after compiling the report)	ActiveReport Script Mapping is required
RPX HTTP Handler (when the *.rpx report is placed on the Web)	ActiveReports 14 RPX Script Mapping is required
RDLX HTTP Handler (when the *.rdlx report is placed on the Web)	ActiveReports 14 RDLX Script Mapping is required

- ParameterReport.aspx: The web form that demonstrates how to generate a report by passing a parameter to the report. This sample uses the Params report from the RpxReports folder of this Sample project. The date list is created by changing the SQL query of the report at run time. In this sample, when the date is selected from the Calendar control, the SQL query is updated and the report is generated. The report is generated dynamically in the SelectedIndexChanged event of the Calendar control. On this form, you can select the Viewer to display the report - **HTMLViewer**, **AcrobatReader**, or **RawHTML**.

  > **Note**: This sample requires write permissions to the ReportOutput folder that is located in the web samples directory.

- Web.config: The configuration file that contains the httpHandlers that allow ActiveReports to process reports on the Web.

  > **Note**: You need to manually update version information here when you update your version of ActiveReports.

- WebControl.aspx: This page is opened by clicking **WebControl for ASP.NET** on the Default.aspx page. By default, it displays the WebViewer control with the Invoice report. In this page, you can select from HTMLViewer,

AcrobatReader, RawHtml viewer types and can also select from Page, Section and RDL report types to be displayed.

Online Samples

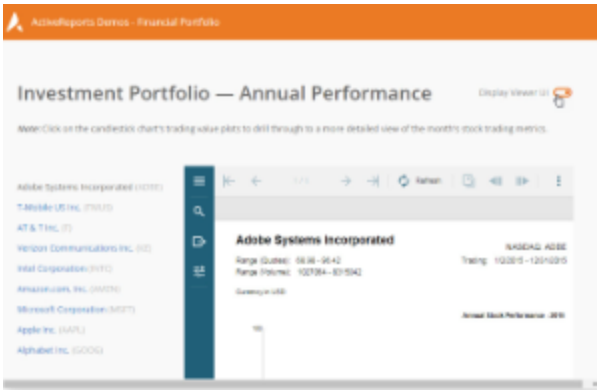
The samples in OnlineSamples folder demonstrate following features. Download these samples from following link: https://github.com/activereports/OnlineSamples14

Online Samples

Sample	Description
Financial Portfolio	This sample demonstrates visualizing stock data with a Candlestick Chart.
Plant Performance	This sample demonstrates how to visualize the Plant Performance KPIs using the Map control.
ReportsGallery_Angular	This sample demonstrates a variety of RDL, Page, and Section reports along with their descriptions.

FinancialPortfolio_Angular

The Financial Portfolio sample demonstrates how to visualize stock data with a Candlestick Chart.



Note: To run this sample, you must have

- Visual Studio 2017 (https://visualstudio.microsoft.com/vs/) version 15.7 or newer.
- .NET Framework Dev Pack (https://dotnet.microsoft.com/download) version 4.6.2 or later.
- Angular requires the Node.js 8.x or 10.x version.

Sample Location

```
..\OnlineSamples14\FinancialPortfolio\
```

Details

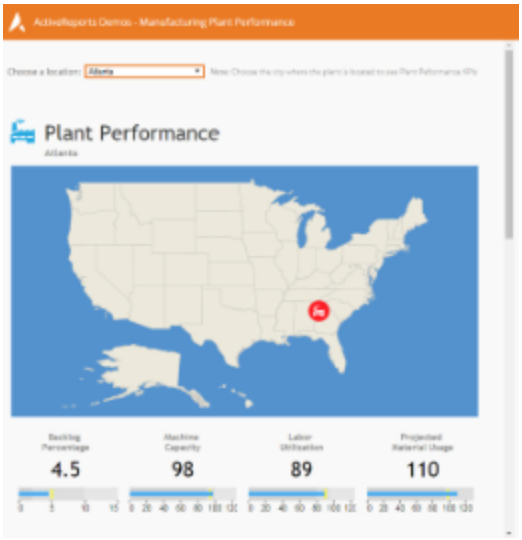
When you run the sample, the start page with the Candlestick Chart appears. In the left panel, you can click a company from the list to see its stock data.

The project consists of the following elements.

- ClientApp folder: This folder contains a standard Angular CLI app that is used for all UI concerns.
- Controllers folder: This folder contains the **ReportsController** files. The **ReportsController** handles the interaction with reports when a report is selected in the left panel.
- Reports folder: The **Reports** folder contains the following reports - **AnnualStockTicker.rdlx** and **MonthlyStockTradingData.rdlx**.
- appsettings.json: The json configuration file.
- readme: This file contains the instructions on how to run the sample project.
- Startup.cs: This is the default startup file.
- Web.config: This configuration file contains the HttpHandlers that allow ActiveReports to process reports on the Web. Note that you need to manually update version information here when you update your version of ActiveReports.

Plant Performance_Angular

The Plant Performance sample demonstrates how to visualize the Plant Performance KPIs using the Map control.



Note: To run this sample, you must have

- Visual Studio 2017 (https://visualstudio.microsoft.com/vs/) version 15.7 or newer.
- .NET Framework Dev Pack (https://dotnet.microsoft.com/download) version 4.6.2 or later.
- Angular requires the Node.js 8.x or 10.x version.

Sample Location

```
..\OnlineSamples14\FinancialPortfolio\
```

Details

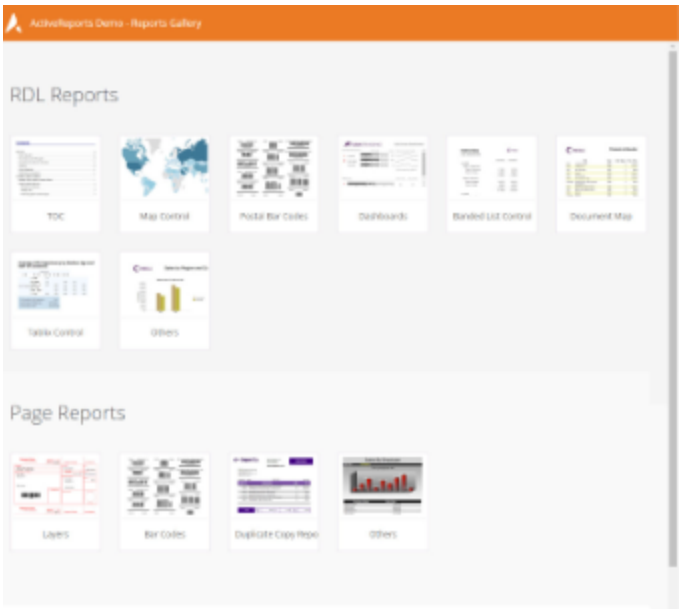
When you run the sample, the start page with the Plant Perfomance KPIs for Atlanta is displayed. On this page, you can choose another location from the drop-down list on top of the page.

The project consist of the following elements.

- ClientApp folder: This folder contains a standard Angular CLI app that is used for all UI concerns.
- Controllers folder: This folder contains the **ReportsController** files. The **ReportsController** handles the interaction with reports when a report is selected in the left panel.
- Reports folder: Contains the **PlantPerformance.rdlx** report.
- appsettings.json: The json configuration file.
- readme: This file contains the instructions on how to run the sample project.
- Startup.cs: This is the default startup file.
- Web.config: This configuration file contains the HttpHandlers that allow ActiveReports to process reports on the Web. Note that you need to manually update version information here when you update your version of ActiveReports.

ReportsGallery_Angular

The Reports Gallery sample demonstrates a list of reports designed using ActiveReports. Use this sample to view examples of WPF spreadsheet applications, and experience Excel-like features such as Excel import or export, charts, themes, and adding skins to spreadsheets.



> **Note**: To run this sample, you must have
>
> - Visual Studio 2017 version 15.7 or newer
> - .NET Framework Dev Pack 4.6.2 or later installed on your machine.
> - Node.js 8.x or 10.x.

Build the sample

1. Start Microsoft Visual Studio and select **File** → **Open** → **Project/Solution**.
2. Go to the sample folder. Double-click the Visual Studio Solution (.sln) file.
3. Right-click the solution in Solution Explorer and select **Restore NuGet Packages**.
4. Press Ctrl+Shift+B, or select **Build** → **Build Solution**.
5. Open Command Prompt and navigate to the sample: ReportsGallery_Angular\ClientApp folder.
6. Run the `npm install` command.

Run the sample

To debug the sample and then run it, press F5 or select **Debug** → **Start Debugging**.

To run the sample without debugging, press Ctrl+F5 or select **Debug** → **Start Without Debugging**.

Walkthroughs

The Walkthroughs section of the User Guide provides you with step-by-step tutorials that you can follow as you create projects in Visual Studio. These walkthroughs cover the key features of Page Reports, RDL Reports and Section Reports in different scenarios. The walkthroughs progress from basic through advanced for Standard and Professional Editions of ActiveReports.

This section contains information about:

Page Report/RDL Report Walkthroughs
: Learn the dynamic features through easy to implement scenarios for Page Report and RDL Report.

Section Report Walkthroughs
: Use the walkthroughs under this section to understand key features of a section report through simple scenarios.

Common Walkthroughs
: Common walkthroughs cover scenarios to introduce the key features of page, RDL and section reports.

Page Report/RDL Report Walkthroughs

Page Report walkthroughs cover scenarios to introduce the key features of Page Reports and RDL reports. Learn about different walkthroughs categorized as follows.

Data
: This section contains the walkthroughs that explain various ways of working with data sources.

Layout
: This section contains the walkthroughs that explain how to create different report layouts.

Chart
: This section contains the walkthrough that demonstrates how to work with the ActiveReports Chart control.

Map
: This section contains the walkthrough that demonstrates how to work with the ActiveReports Map control.

Tablix
: This section contains the walkthroughs that demonstrates how to work with the ActiveReports Tablix data region.

Export
: This section contains the walkthrough that demonstrates how to export your reports into several popular formats like PDF, HTML, Excel, Image and Word.

Preview
: This section contains the walkthrough that demonstrates how to work with custom data and custom code.

Advanced
: This section contains the walkthroughs that demonstrate interactive features of ActiveReports reports.

Data

This section contains the following walkthroughs that fall under the Data category.

Master Detail Reports

This walkthrough demonstrates how to create a master detail report using the Table control and grouping.

Reports with Parameterized Queries

This walkthrough demonstrates how to create a simple dynamic query.

Reports with Stored Procedures

This walkthrough demonstrates how to create a report that uses a stored procedure as a data set.

Reports with XML Data

This walkthrough demonstrates how to connect a report to an XML data source and to create a data set.

Reports with JSON Data

This walkthrough demonstrates how to connect a report to a JSON data source at run time and use a web service to fetch the data with the authorized access.

Reports with CSV Data

This walkthrough demonstrates how to connect a report to a CSV data source.

Expressions in Reports

This walkthrough demonstrates how to use expressions to achieve different effects in a report.

Multiple Datasets in a Data Region

This walkthrough demonstrates how to use multiple datasets in a data region using Lookup function.

Master Detail Reports

You can create a master detail report using the Table control and grouping. The following walkthrough takes you through the step by step procedure of creating a Master Detail report.

The walkthrough is split into the following activities:

- Adding an ActiveReport to a Visual Studio project
- Connecting the report to a data source
- Adding a dataset to the report
- Adding controls to the report to contain data
- Viewing the report

Note:

- This walkthrough uses the **Customer** table from the Reels database. The Reels.mdb file can be downloaded from GitHub: ..\Samples14\Data\Reels.mdb.
- Although this walkthrough uses Page reports, you can also implement this using RDL reports.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout

Run-Time Layout

To add an ActiveReport to the Visual Studio project

1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 Page Report** and in the Name field, rename the file as rptMasterDetail.

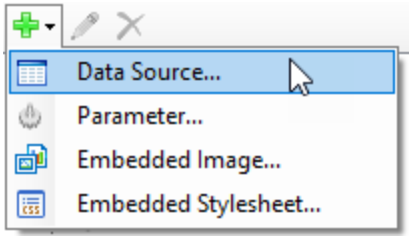
   > **Note**: For a $$**RDL report**$$, in the Add New Item dialog, select **ActiveReports 14 RDL Report**.

4. Click the **Add** button to open a new fixed page report in the designer.

See Quick Start for information on adding different report layouts.

To connect the report to a data source

1. In the Report Explorer, right-click the Data Sources node and select the **Add Data Source** option or select **Data Source** from the Add button.



2. In the Report Data Source Dialog that appears, select the **General** page and in the Name field, enter a name like **ReportData**.
3. On this page, create a connection to the Reels database. See Connect to a Data Source for information on connecting to a data source.

To add a dataset

1. In the Report Explorer, right-click the data source node and select the **Add Data Set** option or select **Data Set** from the Add button.
2. In the DataSet Dialog that appears, select the **General** page and enter the name of the dataset as **CustomerOrders**. This name appears as a child node to the data source icon in the Report Explorer.
3. On the **Query** page of this dialog, in the Query field enter the following SQL query.

SQL Query
`SELECT CustomerID, Title, LastName, Quantity, Price, [Quantity]*[Price] AS Total FROM CustomerOrders WHERE CustomerID < 1010`

4. Click the **Validate DataSet** icon at the top right hand corner above the Query box to validate the query.
5. Click **OK** to close the dialog. Your data set and queried fields appear as nodes in the Report Explorer.

To create a layout for the report

To add a table with grouping to the report

1. From the toolbox, drag a Table data region onto the report design surface and go to the Properties Window to set its **Location** property to 0in, 1in.
2. Click inside the table to display the column and row handles along the top and left sides of the table.
3. To visually group the data within the report, right-click the row handle to the left of the detail row and select **Insert Group.**
4. In the **Table - Groups** dialog that appears, under **Expression** select `=Fields!CustomerID.Value`.
   This groups the details from each customer.
5. Change the **Name** to Customer.

   > **Note:** You cannot change the name of a table group until you have set the expression.
6. Click **OK** to close the dialog. The group header and footer rows are added to the table.

To add a fourth column to the table

1. Select the second column and in the Properties Window, change its **Width** property to **0.92in**.
2. Select the third column and in the Properties Window, change its **Width** property to **1.04in**.
3. Select the first column and in the Properties Window, change its **Width** property to **3.5in**.

   > **Tip:** Making some columns narrower before making other columns wider prevents your report width from changing.
4. Right-click the column handle above the third column and select **Insert Column to the Right**. The inserted fourth column has the same width as the third column, which is **1.04in**.

To add data to the table

1. Place your mouse over the Textbox located in the first column of the group header row of the table to display the field selection adorner.
2. Click the adorner to display the list of available fields from the DataSet and select **LastName**. This automatically places an expression in the group header row and simultaneously places a static label in the table header row.
3. In the Properties window, set the **FontSize** property of this textbox to **12pt**.
4. In the table header row, delete the static label **Last Name**.
5. To display static labels at the beginning of each new group, right-click the row handle to the left of the group header row and select **Insert Row Below**.
6. In the first column of the detail row, use the field selector adorner to select the **Title** field.
7. In the textbox immediately above it, type **Title**.
8. In the table header row, delete the static label **Title**.
9. In the second column of the detail row, use the field selector adorner to select the **Quantity** field. This automatically places an expression in the detail row and simultaneously places a static label in the header row of the same column.
10. In the Properties window, set the **TextAlign** property of the **Quantity** field to **Left**.
11. Cut the static label and paste it into the inserted row immediately above the detail row.
12. In the third column of the detail row, use the field selector adorner to select the **Price** field.
13. In the Properties window, set the following properties.

Property Name	Property Value
TextAlign	Left

Format	C (uses currency formatting)

14. Cut the static label from the group header and paste it into the inserted row immediately above the detail row.
15. Select the **Total** field for the fourth column of the detail row of the table.
16. In the Properties window, set the following properties.

Property Name	Property Value
TextAlign	Left
Format	C (uses currency formatting)

17. Cut the static label from the group header and paste it into the inserted row immediately above the detail row.

To refine the look of the report

1. Right-click any row handle to the left of the table and select **Table Header** to remove the table header row since it is not being used.
2. Select the Header row containing the labels by clicking the table handle to the left of the row.
3. In the Properties window, set the following properties.

Property Name	Property Value
RepeatOnNewPage	True
FontWeight	Bold

4. From the Report Explorer, drag the **Total** field into the group footer row in the fourth column to add subtotaling to the group. Notice that the expression automatically uses the **Sum** function.
5. Go to the Properties window to set the following properties.

Property Name	Property Value
Format	C (uses currency formatting)
FontWeight	Bold

6. From the Report Explorer, drag the **Total** field into the table footer row in the fourth column. This adds grand totaling to the table.
7. Go to the Properties window to set the following properties.

Property Name	Property Value
Format	C (uses currency formatting)
FontWeight	Bold

8. Delete the static label **Total** from the top row.
9. In the Report Explorer select the Table data region and set its FixedSize property to **6.5in, 7in**.

> **Note:** This step is valid only for Page report. For RDL report, follow steps 10-12.

10. For RDL reports, click the gray area below the design surface to give report the focus and from the **Report** menu, select **Page Header**.
11. From the toolbox, drag the TextBox control onto the PageHeader section to span the entire width of the report.
12. Go to the Properties Window to set the following properties.

Property Name	Property Value
TextAlign	Center
FontSize	14pt

Value	Customer Orders

To view the report

- Click the preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

Reports with Parameterized Queries

See Parameterized Reports topic for an alternative approach to build parameterized query for data sets.

The walkthrough is split into the following activities:

- Adding an ActiveReport to a Visual Studio project
- Connecting the report to a data source
- Adding a dataset
- Adding controls to the report to contain data
- Creating a second dataset for use by the parameter list
- Adding parameters to the report
- Changing the Products dataset to use a dynamic query
- Adding a header to display the chosen parameter label
- Viewing the report

Note:

- This walkthrough uses the **MovieType** table from the Reels database. The Reels.mdb file can be downloaded from GitHub: ..\Samples14\Data\Reels.mdb.
- Although this walkthrough uses Page reports, you can also implement this using RDL reports.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout

Run-Time Layout

To add an ActiveReport to the Visual Studio project

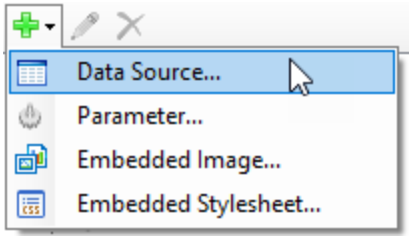
1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.

3. In the Add New Item dialog that appears, select **ActiveReports 14 Page Report** and in the Name field, rename the file as **DynamicQueries**.
4. Click the **Add** button to open a new fixed page report in the designer.

See Quick Start for information on adding different report layouts.

To connect the report to a data source

1. In the Report Explorer, right-click the Data Sources node and select the **Add Data Source** option or select **Data Source** from the Add button.



2. In the Report Data Source Dialog that appears, select the **General** page and in the **Name** field, enter a name like ReportData.
3. On this page, create a connection to the Reels database. See Connect to a Data Source for information on connecting to a data source.

To add a dataset

1. In the Report Explorer, right-click the data source node and select the **Add Data Set** option or select **Data Set** from the Add button.
2. In the DataSet Dialog that appears, select the **General** page and enter the name of the dataset as Products. This name appears as a child node to the data source icon in the Report Explorer.
3. On the **Query** page of this dialog, in the **Query** field enter the following SQL query.

SQL Query

```
SELECT Movie.Title, Product.InStock, Product.StorePrice
FROM MediaType
INNER JOIN
(Movie INNER JOIN
(Product INNER JOIN MovieProduct
  ON Product.ProductID = MovieProduct.ProductID)
 ON Movie.MovieID = MovieProduct.MovieID)
ON MediaType.MediaID = MovieProduct.MediaType
WHERE (((MediaType.MediaID)=1))
ORDER BY Movie.Title
```

4. Click the **Validate DataSet** icon at the top right hand corner above the Query box to validate the query.
5. Click **OK** to close the dialog. Your data set and queried fields appear as nodes in the Report Explorer.

To create a layout for the report

1. From the toolbox, drag a Table data region onto the report design surface and set the following properties in the Properties Window:

Property Name	Property Value
Location	0in, 0.5in

Size	5.5in, 0.75in
FixedSize (only for FixedPageLayout reports)	6.5in, 7in

2. In the Report Explorer from the **Products** dataset, drag the following fields onto the detail row of the table.

Data Field	Column Name
Title	TableColumn1
InStock	TableColumn2
StorePrice	TableColumn3

> **Note:** This automatically places an expression in the detail row and simultaneously places a static label in the header row of the same column.

3. Select the header row, click the table handle to the left of the row and in the Properties Window, set the following properties:

Property Name	Property Value
FontWeight	Bold
BackgroundColor	DarkSeaGreen
RepeatOnNewPage	True

4. Select the **StorePrice** field in the detail row and in the Properties Window, set its **Format** property to Currency.
5. Click the column handle at the top of each column in turn to select it, and in the Properties Window, set the **Width** property as indicated in the table.

Column	Width
First	4.5in
Second	1in
Third	1in

To create a second dataset for use by the parameter list

1. In the Report Explorer, right-click the data source node and select the **Add Data Set** option or select **Data Set** from the Add button.
2. In the DataSet Dialog that appears, select the **General** page and enter the name of the dataset as **MediaType**.
3. On the **Query** page, paste the following SQL command into the **Query** text box:

SQL Query

```
SELECT 0 AS MediaID, "All" AS Description
FROM MediaType
UNION SELECT MediaID, Description
FROM MediaType
ORDER BY Description
```

4. Click the **Validate** icon  to validate the query and to populate the Fields list.
5. Click **OK** to close the dialog. Your data set and queried fields appear as nodes in the Report Explorer.

To add parameters to the report

1. In the Report Explorer, select the Parameters node.
2. Right-click the node and select **Add Parameter** to open the Report - Parameters dialog.
3. In the dialog box that appears, select the parameter from the parameters list.
4. Set properties in the following fields below the parameters list.

   In the **General** tab:

   - Name: MediaType
   - DataType: String
   - Text for prompting users for a value: Select a media type

   In the **Available Values** tab select From query:

   - DataSet: MediaType
   - Value: MediaID
   - Label: Description
5. Click **OK** to close the dialog and add the parameter to the collection. This parameter appears under the Parameters node in the Report Explorer.

To modify the Products dataset to use a dynamic query

1. In the Report Explorer, right-click the **Products** dataset and select **Edit**.
2. In the DataSet dialog that appears, select the **Query** page.
3. In the Query field, change the query to the following expression:

Query
`="SELECT Movie.Title, Product.InStock, Product.StorePrice, MediaType.Description FROM MediaType INNER JOIN (Movie INNER JOIN (Product INNER JOIN MovieProduct ON Product.ProductID = MovieProduct.ProductID) ON Movie.MovieID = MovieProduct.MovieID) ON MediaType.MediaID = MovieProduct.MediaType" & IIf(Parameters!MediaType.Value = 0, ""," WHERE (MediaType = " & Parameters!MediaType.Value & ")") & " ORDER BY Movie.Title"`

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

To add a header to display the chosen parameter label

1. From the toolbox, drag and drop a Textbox control onto the report design surface. In RDL reports, you can place the Textbox control in the PageHeader.
2. Select the Textbox and set the following properties in the Properties window.

Property Name	Property Value
TextAlign	Center
FontSize	14pt
Location	0in, 0in
Size	6.5in, 0.25in
Value	=Parameters!MediaType.Label & " Movies in Stock"

> **Note:** Using **Label** instead of **Value** in the expression displays a more readily understandable Description field instead of the MediaID field we used for the parameter's value.

To view the report

- Go to the preview tab and select a parameter in the Parameters pane to view the report at design time.

OR

- Open the report in the Viewer and select a parameter in the Parameters pane to view the report. See Windows Forms Viewer for further information.

Reports with Stored Procedures

You can create a report using a stored procedure as a dataset. A stored procedure is a group of SQL statements that are used to encapsulate a set of operations or queries to execute on a database.

This walkthrough illustrates how to create a report that uses a stored procedure as a data set. The walkthrough is split into the following activities:

- Adding an ActiveReport to a Visual Studio project
- Connecting the report to a data source
- Adding a dataset (stored procedure) with a parameter
- Creating a layout for the report
- Viewing the report

> **Note:**
>
> - This walkthrough uses a table from the Reels database. The Reels.mdb file can be downloaded from GitHub: ..\Samples14\Data\Reels.mdb.
> - Although this walkthrough uses Page reports, you can also implement this using RDL reports.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout

Run-Time Layout

To add an ActiveReport to the Visual Studio project

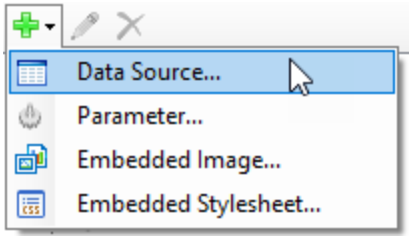
1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.

3. In the Add New Item dialog that appears, select **ActiveReports 14 Page Report** and in the Name field, rename the file as StoredProcedure.
4. Click the **Add** button to open a new fixed page report in the designer.

See Quick Start for information on adding different report layouts.

To connect the report to a data source

1. In the Report Explorer, right-click the Data Sources node and select the **Add Data Source** option or select **Data Source** from the Add button.



2. In the Report Data Source Dialog that appears, select the **General** page and in the Name field, enter a name like Reels.
3. On this page, create a connection to the Reels database. See Connect to a Data Source for information on connecting to a data source.

To add a dataset with a parameter

1. In the Report Explorer, right-click the data source node and select the **Add Data Set** option or select **Data Set** from the Add button.
2. In the DataSet Dialog that appears, select the **General** page and enter the name of the dataset as SalesDataForStore. This name appears as a child node to the data source icon in the Report Explorer.
3. On the **Query** page of this dialog, set the **Command Type** to Stored Procedure.
4. On the **Query** page of this dialog, in the Query field enter the stored procedure name (e.g. SalesDataForStore).
5. Click the **Validate** icon to validate the query. You may receive an error at this point since the required parameters have not yet been added.

6. Go to the **Parameters** page and add a Parameter using the Add(+) button.
7. On the same page, enter **Name** as StoreID and **Value** as 1002**.**
8. Select the **Query** page of **DataSet** dialog, and click the **Validate** icon to validate the query and load the fields.
9. Click **OK** to close the dialog. Your data set and queried fields appear as nodes in the Report Explorer.

To create a layout for the report

1. From the toolbox, drag a Table data region onto the report design surface.
2. In the Table data region, place your mouse over the cells of the table details row to display the field selection adorner.
3. With the Table selected, right-click and open the Properties Window to set the following properties:

Property Name	Property Value
Location	0in, 0.5in
Size	6.5in, 0.75in
FixedSize	6.5in, 7in

4. Click the adorner to show a list of available fields from the SalesDataForStore dataset and add the following fields

to the cells of the table details row.

Cell	Field
Left Cell	StoreID
Middle Cell	UnitsSold
Right Cell	NetSales

This automatically places an expression in the detail row and simultaneously places a static label in the header row of the same column.

5. Select the Header row by clicking the table handle to the left of the row and go to the Properties Window to set the following properties:

Property Name	Property Value
FontWeight	Bold
RepeatOnNewPage	True

6. Click the column handle at the top of each column in turn to select it, and in the Properties Window, set the **Width** property as indicated in the table:

Column	Width
First	3.5in
Second	2in
Third	1in

7. Set the **TextAlign** property of all the columns to Left.
8. From the toolbox, drag the Textbox onto the design surface to span the entire width of the report and go to the Properties Window to set the following properties:

Property Name	Property Value
TextAlign	Center
Size	6.5in, 0.35in
Location	0in, 0.125in
FontSize	14pt
Value	Net Sales by Store

> **Tip**: In a RDL report, you can also add a Page Header to place the Textbox control.

To view the report

- Click the preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

Multiple Datasets in a Data Region

Many a time, we need to display varied data from different datasets into one data region. This is now possible by using the **Lookup** function in a data region.

The **Lookup** function returns a value corresponding to a related or a common field with the same data type in another data set. It is set as an expression in the Value property of a data region's Textbox. The Lookup function in ActiveReports is similar to the Microsoft Excel's VLOOKUP.

Lookup Function

Syntax

=Lookup(<SourceExpression>, <DestinationExpression>, <ResultExpression>, <LookupDataset>)

Parameters

- **Comparison Criteria:** To compare the fields in the Source and the Lookup datasets. The criterion uses only the "=" operator.
- **SourceExpression:** An expression evaluating to a value from the dataset associated with the data region. If this expression is a FieldName, then the value of the Field from the dataset associated with the data region is used.
- **DestinationExpression:** An expression evaluating to a value from the dataset associated with the LookupDataset. If this expression is a FieldName, then the value of the Field from the dataset of the LookupDataset is used.
- **ResultExpression:** An expression evaluating to the Field from the LookupDataset returned by the Lookup function.
- **LookupDataset**: The dataset where the value from the data region's dataset is used to display the related attribute.

Usage

- The data type of the SourceExpression and the DestinationExpression should be same.
- When the Lookup function is used as a value expression in a data region, the expression is evaluated for each row or repeated data of the data region's dataset.
- The Lookup function returns one value if found, and null if no rows are found in the Lookup dataset.
- The Lookup expressions can be a part of aggregated expressions. A user can use the Lookup function in a table group or table header or footer, and sum all values for the table.

Limitations

- Only "=" comparison is supported between SourceExpression and DestinationExpression.
- Non-aggregate expressions such as multiply, mod, AND and OR, are not allowed in the comparison criteria.
- Only one level of Lookup is allowed, that is, nested Lookup functions are not supported.

This walkthrough explains the steps involved in using multiple datasets in a data region. The walkthrough is split into the following activities:

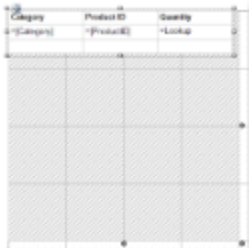
- Adding an ActiveReport to a Visual Studio project
- Connecting the report to a data source
- Adding the datasets
- Adding controls to the report
- Viewing the report

Note:

- This walkthrough uses the SalesResult database. The SalesResult.mdb file can be downloaded from GitHub: ..\Samples14\Data\SalesResult.mdb.
- Although this walkthrough uses Page reports, you can also implement this using RDL reports.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout



Run-Time Layout

To add an ActiveReport to the Visual Studio project

1. Create a new Visual Studio project.
2. In the **New Project** dialog that appears, select **ActiveReports 14 Page Report Application** and in the Name field, rename the file as **SalesResultReport**.
3. Click **OK** to create a new **ActiveReports 14 Page Report Application**. By default a Page report is added to the project.

See Quick Start for information on adding different report layouts.

To connect the report to a data source

1. In the Report Explorer, right-click the Data Sources node and select the **Add Data Source** option or select **Data Source** from the Add button.
2. In the Report Data Source Dialog that appears, select the **General** page and in the Name field, enter a name like SalesResultData.
3. On this page, create a connection to the SalesResult database. See Connect to a Data Source for information on connecting to a data source.

To add the datasets

To add Dataset1

1. In the Report Explorer, right-click the data source node and select the **Add Data Set** option or select **Data Set** from the Add button.
2. In the DataSet Dialog that appears, select the **General** page and let the name of the dataset be **Dataset1**. This name appears as a child node to the data source icon in the Report Explorer.
3. On the **Query** page of this dialog, in the **Query** field enter the following SQL query.

SQL Query
`SELECT M01Product.Category, M01Product.ProductID`

```
FROM M01Product
```

4. Click the **Validate DataSet** icon at the top right hand corner above the Query box to validate the query.
5. Click **OK** to close the dialog. Your data set and queried fields appear as nodes in the Report Explorer.

   The Dataset1 contains following fields:

   - Category
   - ProductID

To add Dataset2

1. Repeat Steps 1 and 2 to add another dataset with name **Dataset2.**
2. On the **Query** page of this dialog, in the **Query** field enter the following SQL query.

SQL Query
`SELECT * FROM T01Result`

3. Click the **Validate DataSet** icon at the top right hand corner above the Query box to validate the query.
4. Click **OK** to close the dialog. Your data set and queried fields appear as nodes in the Report Explorer.

   The Dataset2 contains following fields:

   - ID
   - ProductID
   - Quantity
   - PDate
   - FY

To add controls to the report

1. From the toolbox, drag a Table data region onto the design surface of the report.
2. Go to the Properties Window to set the properties of Table data region as follows:

Property Name	Property Value
FixedSize	4in, 4in
Location	0in, 0in
Size	3.875in, 0.75in
DataSetName	Dataset1

3. Hover your mouse over the text boxes of the Table Details row to access the field selection adorner and set the following fields in the table cells along with their properties.

Cell	Field
TextBox4	Category
TextBox5	ProductID

   This automatically places an expression in the details row and simultaneously places a static label in the header row of the same column.

4. Select TextBox6 of the Table data region and from the Properties pane, set the following properties:

Property Name	Property Value

Value	=Lookup(Fields!ProductID.Value, Fields!ProductID.Value, Fields!Quantity.Value, "DataSet2")
TextAlign	Left

The expression in the Value property returns the value of Quantity from Dataset2, corresponding to the related data field ProductID in Dataset1.

5. Select TextBox3 of the Table data region and from the Properties pane, set the following properties:

Property Name	Property Value
Value	Quantity
TextAlign	Left

6. Select the header row using the row handle to the left and in the Properties Window, set the **FontWeight** property to **Bold**.

To view the report

- Click the preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

Layout

This section contains the following walkthroughs that fall under the Layout category.

BandedList Reports
: This walkthrough demonstrates how to create a grouped BandedList report.

Collate Multiple Copies of a Report
: This walkthrough demonstrates how to use the collation feature in a report that contains layouts on two page tabs.

Columnar Layout Reports (RDL)
: This walkthrough demonstrates how to create a RDL report using columns.

Overflow Data in a Single Page(Page Report)
: This walkthrough demonstrates how to use the OverflowPlaceHolder controls at any location on the same page to build a rich report layout.

Overflow Data in Multiple Pages(Page Report)
: This walkthrough demonstrates how to create two different layouts using the OverflowPlaceHolder control.

Recursive Hierarchy Reports
: This walkthrough demonstrates how to create a report using a recursive hierarchy and the Level function to show parent-child relationships in data.

Single Layout Reports
: This walkthrough demonstrates how to create a layout at design time on a single page tab and apply it to the entire report.

Subreports in Page/RDL Reports
: This walkthrough demonstrates how to create a report using a subreport.

Banded List Reports

You can create a freeform report with groups using the Banded List control. This walkthrough illustrates how to create a grouped BandedList report.

The walkthrough is split up into the following activities:

- Adding an ActiveReport to a Visual Studio project
- Connecting the report to a data source
- Adding a dataset
- Adding a banded list with grouping
- Adding controls to the banded list
- Viewing the report

> **Note:**
>
> - This walkthrough uses the **Movie** table from the Reels database. The Reels.mdb file can be downloaded from GitHub: ..\Samples14\Data\Reels.mdb..
> - Although this walkthrough uses Page reports, you can also implement this using RDL reports.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout

Run-Time Layout

To add an ActiveReport to the Visual Studio project

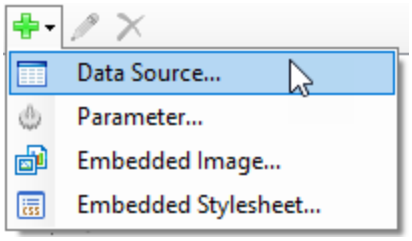
1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 Page Report** and in the Name field, rename the file as rptBandedList.
4. Click the **Add** button to open a new fixed page report in the designer.

See Quick Start for information on adding different report layouts.

To connect the report to a data source

1. In the Report Explorer, right-click the Data Sources node and select the **Add Data Source** option or select **Data**

Source from the Add button.



2. In the Report Data Source Dialog that appears, select the **General** page and in the Name field, enter a name like ReportData.
3. On this page, create a connection to the Reels database. See Connect to a Data Source for information on connecting to a data source.

To add a dataset

1. In the Report Explorer, right-click the data source node and select the **Add Data Set** option.
2. In the DataSet Dialog that appears, select the **General** page and enter the name of the dataset as Movies. This name appears as a child node to the data source icon in the Report Explorer.
3. On the **Query** page of this dialog, change the Command Type to **TableDirect** and enter Movie into the Query text box.
4. Click the **Validate DataSet** icon at the top right hand corner above the Query box to validate the query.

5. Click **OK** to close the dialog. Your data set and queried fields appear as nodes in the Report Explorer.

To add the BandedList with grouping

1. From the toolbox, drag a BandedList data region onto the design surface and go to the properties window to set the **DataSetName** property to Movies.

   > **Note**: To select the banded list, click inside any band in the list and click the four-way arrow that appears at the top left of the control.

2. Right-click inside the banded list and select **Footer**. This removes the footer band that we are not using in this walkthrough.
3. Right-click inside the banded list and select **Insert Group**. This adds a group header between the BandedList header and the detail section, and a group footer below the detail section, and opens the **BandedList - Groups** dialog.
4. In the BandedList - Groups dialog, drop down the list of expressions in the **Group on** expression box and select =Fields!YearReleased.Value to group the movies based on the year in which they were released.
5. In the same dialog, change the **Name** to Year.
6. In the same dialog under **Layout**, clear the **Include group footer** checkbox to remove the group footer.
7. Click the **Add** icon. This adds a second group header under the first, and a group footer below the detail section.
8. In the same dialog, select the newly added group from the list of groups, drop down the list of expressions in the **Group on** expression box and select =Fields!MPAA.Value to group the movies based on the rating (for example, G, PG, etc).
9. In the same dialog, change the **Name** to MPAA.
10. In the same dialog under **Layout**, clear the **Include group footer** checkbox to remove the group footer.
11. Click **OK** to close the dialog.

To add controls to the BandedList

1. From the toolbox, drag a TextBox control onto the top-most band (BandedList1_Header) and in the Properties

window, set the following properties:

Property Name	Property Value
BackgroundColor	Gray
Color	White
FontSize	14pt
FontWeight	Bold
Location	0in, 0in
Size	6.5in, 0.25in
TextAlign	Center
Value	Movie Details

2. Click the **BandedList1_Header** band adorner (the grey bar along the top of the band) to select the banded list header.
3. In the Properties window, set the following properties for the banded list header:

Property Name	Property Value
RepeatOnNewPage	True
Height	0.25in

4. From the Report Explorer drag the **YearReleased** field onto the first grouping band (Year_Header) of the banded list.
5. With this field selected, go to the Properties window to set the following properties:

Property Name	Property Value
BackgroundColor	DarkGray
FontWeight	Bold
Location	0in, 0in
Size	6.5in, 0.25in
TextAlign	Left
Value	=First(Fields!YearReleased.Value)

6. Click the **Year_Header** band adorner (the grey bar along the top of the band) to select it and go to the properties window to set the **Height** property of the header band to 0.25in.
7. From the Report Explorer, drag the **MPAA** field into the second grouping band (MPAA_Header) of the banded list.
8. Go to the Properties window to set the following properties:

Property Name	Property Value
BackgroundColor	Silver
BorderColor	Gray
BorderStyle	Solid
FontWeight	Bold
Location	0in, 0in

Size	6.5in, 0.25in

> **Note:** The **Value** property is automatically set to an expression using the **First** aggregate. This displays the movie rating for each group.

9. Click the **MPAA_Header** band adorner (the grey bar along the top of the band) to select it and go to the properties window to set the **Height** property of the header band to **0.25in**.
10. Click inside the detail band to select it and in the properties window, set its **Height** property to **1.2in**.
11. From the Report Explorer, drag the following six fields onto the detail band and in the properties window, set their properties as indicated.

Data Field	Property Name
Title	Location: 0in, 0.25in Size: 3.75in, 0.25in
Country	Location: 1.25in, 0.5in Size: 1in, 0.25in
Language	Location: 1.25in, 0.75in Size: 1in, 0.25in
Length	Location: 5.375in, 0in Size: 1in, 0.25in TextAlign = Left
UserRating	Location: 5.375in, 0.5in Size: 1in, 0.25in TextAlign = Left
IsColor	Location: 5.375in, 0.75in Size: 1in, 0.25in

> **Note:** When you drag and drop fields from a dataset in the Report Explorer onto the design surface, these fields are automatically converted to Textbox controls that you can modify by setting the control properties in the Properties Window.

12. Once you've arranged the fields, select the **IsColor** field and in the properties window, change the **Value** property to the following expression so that instead of "True" or "False," it will read "Color" or "Black and White."
```
=Iif(Fields!IsColor.Value=True, "Color", "Black and White")
```
13. From the toolbox, drag six TextBox controls onto the detail band and in the properties window, set their properties as indicated.

TextBox1

Property Name	Property Value
Value	Movie Title:
Location	0in, 0in
Size	1in, 0.25in
FontWeight	Bold

TextBox2

Property Name	Property Value
Value	Country of origin:
Location	0in, 0.5in
Size	1.25in, 0.25in
FontWeight	Bold

TextBox3

Property Name	Property Value
Value	Language:
Location	0in, 0.75in
Size	1in, 0.25in
FontWeight	Bold

TextBox4

Property Name	Property Value
Value	Length:
Location	4.25in, 0in
Size	1in, 0.25in
FontWeight	Bold
TextAlign	Right

TextBox5

Property Name	Property Value
Value	User Rating:
Location	4.25in, 0.5in
Size	1in, 0.25in
FontWeight	Bold
TextAlign	Right

TextBox6

Property Name	Property Value
Value	Format:
Location	4.25in, 0.75in

Size	1in, 0.25in
FontWeight	Bold
TextAlign	Right

14. From the toolbox, drag a Line control onto the detail band and in the Properties window, set the properties.

Property Name	Property Value
LineColor	Gray
LineWidth	3pt
Location	0in, 1.12in
EndPoint	6.5in, 1.12in

15. For a Page report, in the Report Explorer select the BandedList control and in the Properties window, set its FixedSize property to **6.5in, 7in**.

To view the report

- Click the preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

Collate Multiple Copies of a Report

In a Page Report you can create reports with multiple themes, where you can control the page order of a rendered report by selecting the collation mode. This walkthrough uses a report that contains layouts on two page tabs in order to illustrate the collation feature.

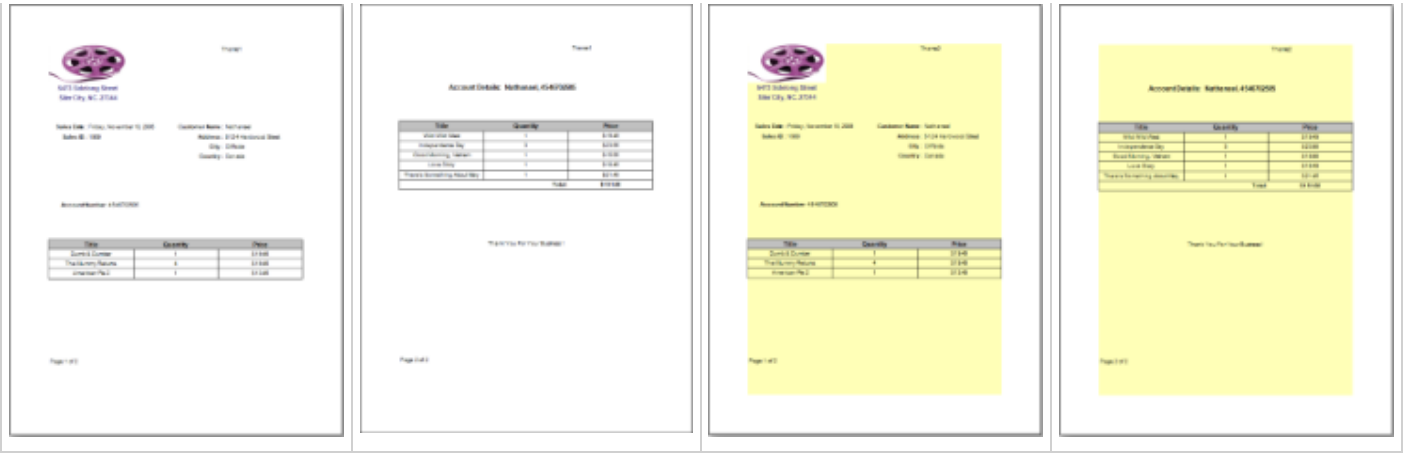
This walkthrough is split into the following activities:

- Creating report layout on multiple pages
- Adding themes
- Applying themes
- Setting up collation
- Viewing the report

> **Note:**
>
> - This walkthrough uses the **CustomerOrders** table from the Reels database. The Reels.mdb file can be downloaded from GitHub: ..\Samples14\Data\Reels.mdb.
> - Although this walkthrough uses Page reports, you can also implement this using RDL reports.

When you complete this walkthrough you get a report that looks similar to the following. These images show the result of the **Value** collation mode.

Page 1	Page 2	Page 3	Page 4



To create a report layout on multiple pages

In a Page Report, you can apply more than one layout to a single report by using page tabs. See the walkthrough Overflow Data in Multiple Pages, to learn how to create a report that consists of a first page with a different layout from all subsequent pages.

> **Note:** The information provided henceforth is an extension of the Overflow Data in Multiple Pages walkthrough, so we recommend that you go through this topic before moving to the next procedure.

To add themes to the report

1. In the Designer, click the gray area around the report page to select a report.
2. In the Properties window, select the **Themes** property and click the ellipsis (...) button to open the **Report - Themes** dialog.
3. In the **Report - Themes** dialog that opens, click the **New** icon above the list of themes.
4. In the Theme Editor that opens, define the colors and constant expressions for your new theme under the corresponding tabs as follows:

   Theme 1

Property Name	Property Value	Description
Text/Background - Light1 (**Colors** tab)	Default (white)	To set the background color for the theme.
Name (**Constants** tab)	Constant1	To define a name for the constant expression to be used within in a theme.
Value (**Constants** tab)	Theme1	To associate a value to be used as a constant in the expression.
Description (**Constants** tab)	Default Theme	To describe a constant expression.

5. In the Theme Editor, click **OK** and in the Save As dialog that opens, choose a directory on your local machine and enter the name, Default Theme for your new theme.
6. Click **Save** to save the theme at the desired location.
7. Repeat steps 3-6 above to add another theme to the report. Define the colors and constant expressions of the theme under the corresponding tabs as follows:

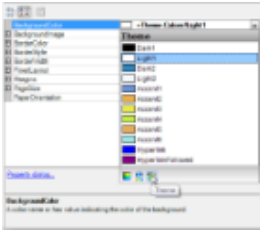
Theme 2

Property Name	Property Value	Description
Text/Background - Light1 (**Colors** tab)	Yellow	To set the background color for the theme.
Name (**Constants** tab)	Constant1	To define a name for the constant expression to be used within in a theme.
Value (**Constants** tab)	Theme2	To associate a value to be used as a constant in the expression.
Description (**Constants** tab)	Yellow Theme	To describe a constant expression.

To apply themes to the report

In order to apply themes to a report, you need to set the theme as a value of a property. In this walkthrough we set the theme in the BackgroundColor property.

1. In the Report Explorer, select the **Page 1** node to open the layout for the first page of the report.
2. In the Properties window, go to the **BackgroundColor** property and click the arrow to display the drop-down list of values.
3. In the list that appears, go to the **Theme** button and select **Light1**.



4. In the Report Explorer, select **Page 2** node to open the layout for the second page of the report.
5. In the Properties window, go to the **BackgroundColor** property and click the arrow to display the drop-down list of values.
6. In the list that appears, go to the **Theme** button and select **Light1**.
7. From the Visual Studio toolbox, **ActiveReports 14 Page Report** tab, drag the Textbox control and drop it onto the Page 1 design surface.
8. With this TextBox control selected, set the following properties in the properties window. This is to display a constant value associated with the Theme.

Property Name	Property Value
Location	4.4in, 0in
Size	2.1in, 0.25in
Value	= Theme.Constants("Constant1") **Note:** To set this value, enter it in the Value field of the Properties window. You may also use the constant expressions (see Themes in Expression Editor that appears when you click the ellipses (...) button adjacent to the Value property.

9. From the Visual Studio toolbox, drag the Textbox control and drop it onto the Page 2 design surface. Set the properties specified in step 8 above in the properties window.

Your layout now contains two themes and a constant expression displaying the theme you are using on the top left corner of the page.

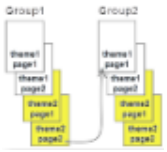
To set up collation

Collation is set to determine the order in which the report pages are rendered when you have multiple themes in the report. In order to define the order, you need to set the **CollateBy** property.

1. In the Designer, click the gray area around the report page to select the report.
2. In the Properties Window, go to the **CollateBy** property and select the **Simple** mode. The report will be rendered in the following way - the report renders all pages with theme 1, then all pages with theme 2.

3. In the Properties Window, go to the **CollateBy** property and select the **ValueIndex** mode. The report will be rendered by page number. For example, if you have a report with 2 themes, the report renders page 1 for theme 1 and 2, then page 2 for theme 1 and 2, and so on.

4. In the Properties Window, go to the **CollateBy** property and select the **Value** mode. The report will be rendered by the grouping expression that you specify in the FixedPage dialog. For example, if you have a report comprising of 2 themes with grouping, the report renders group1 (pages 1 and 2 of theme1, then pages 1 and 2 of theme2), then group 2 (pages 1 and 2 of theme1, then pages 1 and 2 of theme2), and so on.



To view the report

- Click the preview tab to view the report.

OR

- See Windows Forms Viewer to display report in the Viewer at run time.

Columnar Layout Reports (RDL)

In RDL report, you can create a columnar report layout by using the **Columns** property of the report. This walkthrough illustrates how to create a RDL report using columns, and is split up into the following activities:

- Adding an ActiveReport to a Visual Studio project

- Connecting to a the data source
- Adding a dataset
- Creating a column report layout
- Viewing the report

> **Note:** This walkthrough uses the **CustomerMailingList** table from the Reels database. The Reels.mdb file can be downloaded from GitHub: ..\Samples14\Data\Reels.mdb.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout

Run-Time Layout

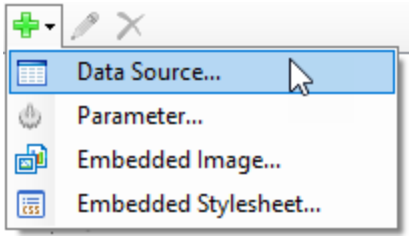
To add an ActiveReport to the Visual Studio project

1. Create a new Visual Studio project.
2. From the Project menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 RDL Report** and in the Name field, rename the file as rptRDLColumnLayout.
4. Click the **Add** button to open a new RDL report in the designer.

See Quick Start for information on adding different report layouts.

To connect the report to a data source

1. In the Report Explorer, right-click the Data Sources node and select the **Add Data Source** option or select **Data Source** from the Add button.



2. In the Report Data Source Dialog that appears, select the **General** page and in the Name field, enter a name like ReportData.
3. On this page, create a connection to the Reels database. See Connect to a Data Source for information on connecting to a data source.

To add a dataset

1. In the Report Explorer, right-click the data source node and select the **Add Data Set** option or select **Data Set** from the Add button.
2. In the DataSet Dialog that appears, select the **General** page and enter the name of the dataset as CustomerList. This name appears as a child node to the data source icon in the Report Explorer.
3. On the **Query** page of this dialog, in the **Query** field enter the following SQL query.

SQL Query

```
SELECT TOP 100 * FROM CustomerMailingList
UNION
SELECT TOP 100 * FROM CustomerMailingList WHERE Country = "USA"
ORDER BY 8 DESC
```

4. Click the **Validate DataSet** icon at the top right hand corner above the Query box to validate the query.

5. Click **OK** to close the dialog. Your data set and queried fields appear as nodes in the Report Explorer.

To create a column layout for the report

1. In the Report Explorer, select **Body** and set the following properties in the properties window.

Property Name	Property Value
Columns	2
ColumnSpacing	0.25in
Size	2.625in, 1in

2. In the Visual Studio toolbox, go to the **ActiveReports 14 Page Report** tab and drag the List data region onto the design surface.
3. In the Properties Window, set the following properties for the List.

Property Name	Property Value
DataSetName	CustomerList
Size	2.5in, 1in

4. In the Visual Studio toolbox, go to the **ActiveReports 14 Page Report** tab and drag three TextBox controls onto the List data region added above.
5. In the Properties Window, set the following properties for **TextBox1**.

Property Name	Property Value
Location	0in, 0in
Size	2.5in, 0.25in
DataElementName	FirstName
Name	FirstName
Value	=Fields!FirstName.Value & IIF(Fields!MiddleInitial.Value Is Nothing, "", " " & Fields!MiddleInitial.Value) & " " & Fields!LastName.Value
CanGrow	False

6. In the Properties Window, set the following properties for **TextBox2**.

Property Name	Property Value
Location	0in, 0.25in
Size	2.5in, 0.25in
DataElementName	CustomerAddress1
Name	CustomerAddress1
Value	=Fields!Address1.Value & IIF(Fields!Address2.Value is Nothing, "", vbCrLf & Fields!Address2.Value)
CanGrow	False
CanShrink	True

7. In the Properties window, set the following properties for **TextBox3**.

Property Name	Property Value
Location	0in, 0.50in
Size	2.5in, 0.25in
DataElementName	CustomerCity
Name	CustomerCity
Value	=Fields!City.Value & ", " & Fields!Region.Value & " " & Fields!PostalCode.Value & " " & IIf(Fields!Country.Value = "USA", "", Fields!Country.Value)
CanGrow	False

To view the report

- Click the preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

If you want to create a similar layout in Page report, see Overflow Data in a Single Page(Page Report).

Overflow Data in a Single Page(Page Report)

In a Page Report, ActiveReports allows you to arrange the OverflowPlaceHolder controls at any location on the same page along with the host data region to build a rich report layout. The following walkthrough demonstrates this by using a Table data region and three OverflowPlaceHolder controls on a single page to create a columnar display.

This walkthrough is split into the following activities:

- Adding an ActiveReport to a Visual Studio project
- Connecting to a data source

- Adding a dataset
- Creating a columnar layout
- Viewing the report

> **Note:** This topic uses the **Movie** table in the Reels database. The Reels.mdb file can be downloaded from GitHub: ..\Samples14\Data\Reels.mdb.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout

Notice that the run time report layout below is similar to the one you see at design time except for the data which you see at run time or when you preview the report.

Run-Time Layout

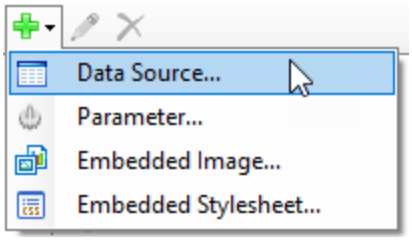
To add an ActiveReport to the Visual Studio project

1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 Page Report** and in the Name field, rename the file as rptColumnarLayout.
4. Click the **Add** button to open a new fixed page report in the designer.

See Quick Start for information on adding different report layouts.

To connect the report to a data source

1. In the Report Explorer, right-click the Data Sources node and select the **Add Data Source** option or select **Data Source** from the Add button.



2. In the Report Data Source Dialogthat appears, select the **General** page and in the Name field, enter a name like ColumnData.
3. On this page, create a connection to the **Reels** database. See Connect to a Data Source for information on connecting to a data source.

To add a dataset

1. In the Report Explorer, right-click the data source node and select the **Add Data Set** option or select **Data Set** from the Add button.
2. In the DataSet Dialog that appears, select the **General** page and enter the name of the dataset as MovieList. This name appears as a child node to the data source icon in the Report Explorer.
3. On the **Query** page of this dialog, in the **Query** field enter the following SQL query.

SQL Query
`SELECT * FROM Movie`

4. Click the **Validate DataSet** icon at the top right hand corner above the Query box to validate the query.

5. Click **OK** to close the dialog. Your data set and queried fields appear as nodes in the Report Explorer.

To create a layout for the report

1. In the Visual Studio toolbox, go to **ActiveReports 14 Page Report** tab and drag a TextBox control onto the design surface.
2. Select the TextBox control and go to the properties window to set the following properties. This TextBox functions as the title in the report layout.

Property Name	Property Value
Location	0in, 0in
BackgroundColor	DarkCyan
Font	Normal, Arial, 20pt, Bold
Size	6.5in, 0.5in
TextAlign	Center
Value	Movie Database
VerticalAlign	Middle

3. From the Visual Studio toolbox, drag and drop a Table data region onto the design surface and set its following properties in the properties window.

Property Name	Property Value

Location	0.125in, 1in
BackgroundColor	Azure
RepeatHeaderOnNewPage	True
Size	3in, 0.66in
FixedSize	3in, 3.5in
OverflowName	OverflowPlaceHolder1 **Note:** Set this property after you add the OverflowPlaceHolder1 to the design surface in the next step, to handle the data that exceeds the fixed size of the Table data region.

4. In the Table data region, right click the footer row and select **Table Footer** from the context menu to delete the unnecessary footer row.
5. In the table details row, add the following fields using the field selection adorner:

Cell	Field Name
Left	Title
Center	YearReleased
Right	UserRating

6. In the table details row, select the left cell containing the Title field and go to the Properties window to set the **ShrinkToFit** property to **True**. This will avoid clipping of longer movie titles and fit the string in the same cell in a smaller font size.
7. In the design view, select the following rows of the Table data region by clicking the table handle to the left of the row and go to the Properties window to set the following properties:

Row	Property Name
TableHeader	BackgroundColor: PaleTurquoise BorderStyle: Solid FontSize: 10pt FontWeight: Bold TextAlign: Left
TableDetail	BorderStyle: Solid TextAlign: Left FontSize: 9pt

8. From the Visual Studio toolbox, drag and drop three OverflowPlaceHolder controls onto the design surface and the set the following properties in the Properties window for each of them so that the data appears in a columnar format.

Control Name	Property
OverflowPlaceHolder1	Location: 3.375in, 1in

	Size: 3in, 3.5in OverflowName: OverflowPlaceHolder2 **Note:** Set this property after you add the OverflowPlaceHolder2 to the design surface.
OverflowPlaceHolder2	Location: 0.125in, 5in Size: 3in, 3.5in OverflowName: OverflowPlaceHolder3 **Note:** Set this property after you add the OverflowPlaceHolder3 to the design surface.
OverflowPlaceHolder3	Location: 3.375in, 5in Size: 3in, 3.5in

Note: Notice that the OverflowName property is set to link each OverflowPlaceHolder control to the next one.

To view the Report

- Click the preview tab to view the report.

OR

- See Windows Forms Viewer to display report in the Viewer at run time.

Overflow Data in Multiple Pages(Page Report)

In a Page Report you can create different layouts in a single report by designing your report on more than one page tab. This walkthrough describes the steps to create two different layouts and how data flows from the first layout to the next using the Overflow Place Holder control.

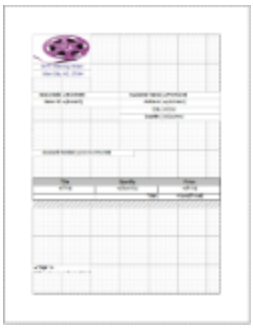
This walkthrough is split into the following activities:

- Adding an ActiveReport to a Visual Studio project
- Connecting to a the data source
- Adding a dataset
- Creating a layout for the first page
- Creating a layout for subsequent pages
- Viewing the Report

Note: This walkthrough uses the **CustomerOrders** table from the Reels database. The Reels.mdb file can be downloaded from GitHub: ..\Samples14\Data\Reels.mdb.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout

Page 1	Page 2
	

Notice that the run-time report layout below is similar to the one you see at design time except for the data which is added to the report at run time or when you preview it.

Run-Time Layout

Page 1	Page 2
	

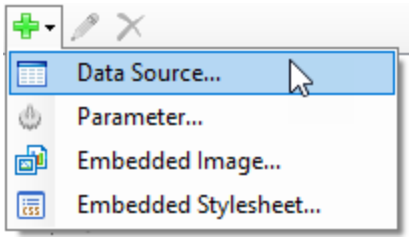
To add an ActiveReport to the Visual Studio project

1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 Page Report** and in the Name field, rename the file as rptMultipleLayout.
4. Click the **Add** button to open a new fixed page report in the designer.

See Quick Start for information on adding different report layouts.

To connect the report to a data source

1. In the Report Explorer, right-click the Data Sources node and select the **Add Data Source** option or select **Data Source** from the Add button.



2. In the Report Data Source Dialog that appears, select the **General** page and in the Name field, enter a name like CustomerData.
3. On this page, create a connection to the Reels database. See Connect to a Data Source for information on connecting to a data source.

To add a dataset

1. In the Report Explorer, right-click the data source node and select the **Add Data Set** option or select **Data Set** from the Add button.
2. In the DataSet Dialog that appears, select the **General** page and enter the name of the dataset as OrdersList. This name appears as a child node to the data source icon in the Report Explorer.
3. On the **Query** page of this dialog, in the **Query** field enter the following SQL query.

SQL Query
`Select * from customerorders`

4. Click the **Validate DataSet** icon at the top right hand corner above the Query box to validate the query.
5. Click **OK** to close the dialog. Your data set and queried fields appear as nodes in the Report Explorer.

To create a layout for the first page

1. Click the design surface to select the page, go to the properties window and set the **FixedLayout - Grouping** property to `=Fields!SalesID.Value`. See Group Data to understand grouping further.
2. In the Report Explorer, right-click the Embedded Images node to select **Add Embedded Image**.
   OR
   Alternatively, you can also select the **Embedded Images** option by clicking the Add (+) icon at the top on the Report Explorer.
   This opens the **Open** dialog where you can select an image to embed in the report.
3. Drag and drop the embedded image, which now appears as a node in the Report Explorer, onto the Page 1 design surface and in the properties window set its **Location** property to 0in, 0in.
4. From the Visual Studio toolbox, drag the following controls from the **ActiveReports 14 Page Report** tab, drop it onto the Page 1 design surface and set the following properties in the properties window.

Controls

Control	Properties
Textbox	Color: DarkSlateBlue Font: Normal, Arial, 11pt, Bold Location: 0in, 1in Size: 2in, 0.25in TextAlign: Center Value: 5473 Sidelong Street
Textbox	Color: DarkSlateBlue

	Font: Normal, Arial, 11pt, Bold Location: 0in, 1.25in Size: 2in, 0.25in TextAlign: Center Value: Siler City, NC. 27344
Textbox	Font: Normal, Arial, 10pt, SemiBold Location: 0in, 2in Size: 1in, 0.25in TextAlign: Right Value: Sales Date :
Textbox	Font: Normal, Arial, 10pt, SemiBold Location: 0in, 2.25in Size: 1in, 0.25in TextAlign: Right Value: Sales ID :
Textbox	Font: Normal, Arial, 10pt, SemiBold Location: 3in, 2in Size: 1.5in, 0.25in TextAlign: Right Value: Customer Name :
Textbox	Font: Normal, Arial, 10pt, SemiBold Location: 3in, 2.25in Size: 1.5in, 0.25in TextAlign: Right Value: Address :
Textbox	Font: Normal, Arial, 10pt, SemiBold Location: 3in, 2.50in Size: 1.5in, 0.25in TextAlign: Right Value: City :
Textbox	Font: Normal, Arial, 10pt, SemiBold Location: 3in, 2.75in Size: 1.5in, 0.25in TextAlign: Right Value: Country :
Textbox	Font: Normal, Arial, 10pt, SemiBold Location: 0in, 4in Size: 1.5in, 0.25in TextAlign: Right Value: Account Number:
Table	BorderStyle: Solid FixedSize: 6.5in, 1in Location: 0in, 5in OverflowName: OverflowPlaceHolder1

> **Note:** Set this property after you add the OverflowPlaceHolder1 control to the design surface to handle the data that exceeds the fixed size of the Table data region.

RepeatHeaderOnNewPage: True
Size: 6.5in, 0.75in

5. From the Report Explorer drag and drop the following fields onto the Page 1 design surface and set the following properties in the properties window.

Fields

Field	Properties
SaleDate	Format: D Location: 1in, 2in Size: 2in, 0.25in TextAlign: Left
SalesID	Location: 1in, 2.25in Size: 2in, 0.25in TextAlign: Left
FirstName	Location: 4.5in, 2in Size: 2in, 0.25in TextAlign: Left
Address1	Location: 4.5in, 2.25in Size: 2in, 0.25in TextAlign: Left
City	Location: 4.5in, 2.5in Size: 2in, 0.25in TextAlign: Left
Country	Location: 4.5in, 2.75in Size: 2in, 0.25in TextAlign: Left
AccountNumber	Location: 1.5in, 4in Size: 2in, 0.25in TextAlign: Left

6. Hover your mouse over the columns of the Table Details row to access the field selection adorner and set the following fields in the table cells along with their properties.

Cell	Field	Properties
Left Cell	Title	BorderStyle: Solid TextAlign: Center
Middle Cell	Quantity	BorderStyle: Solid TextAlign: Center
Right Cell	Price	BorderStyle: Solid

		Format: c TextAlign: Center

This automatically places an expression in the details row and simultaneously places a static label in the header row of the same column.

7. Select the Table Header row and set the following properties in the properties window.

Property Name	Value
BackgroundColor	Silver
Font	Normal, Arial, 11pt, Bold
RepeatOnNewPage	True
TextAlign	Center

8. Set the properties of the following cells on the Table Footer row through the properties window.

Cell	Properties
Middle Cell	Font: Normal, Arial, 10pt, Bold TextAlign: Right Value: Total:
Right Cell	Font: Normal, Arial, 10pt, Bold Format: c TextAlign: Center Value: `=Sum(Fields!Price.Value)`

9. In the Report Explorer, expand the Common Values node and drag and drop the **Page N of M (Section)** field onto the Page 1 design surface and set its **Location** property to 0in, 8in in the properties window.

To create a layout for subsequent Pages

1. Click the **New** tab to add a new page to the report layout. This page is named **Page 2** by default.
2. From the Visual Studio toolbox, drag the following controls from the **ActiveReports 14 Page Report** tab, drop it onto the Page 2 design surface and set the properties in the properties window.

Controls

Control	Properties
Textbox	Font: Normal, Arial, 12pt, Bold Location: 0in, 1in Size: 2.625in, 0.25in TextAlign: Right Value: Account Details:
Textbox	Font: Normal, Arial, 12pt, Bold Location: 2.625in, 1in Size: 3.25in, 0.25in Value: `=Fields!FirstName.Value + ", " +`

	`Fields!AccountNumber.Value`
OverflowPlaceHolder	Location: 0in, 2in Size: 6.5in, 2in
Textbox	Location: 1.75in, 5in Size: 3in, 0.25in TextAlign: Center Value: Thank You For Your Business!

3. In the Report Explorer, expand the Common Values node and drag and drop the **Page N of M (Section)** field onto the Page 2 design surface and set its **Location** property to 0in, 8in in the properties window.

To view the report

- Click the preview tab to view the report.

OR

- See Windows Forms Viewer to display report in the Viewer at run time.

Recursive Hierarchy Reports

You can create a report using a recursive hierarchy and the Level function to show parent-child relationships in data. This walkthrough illustrates how to create a recursive hierarchy report.

The walkthrough is split into the following activities:

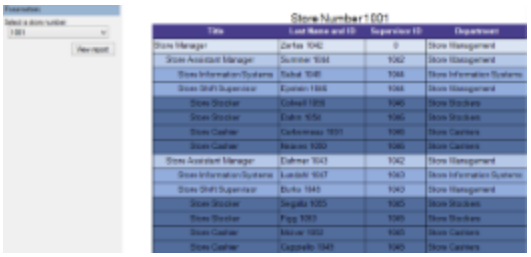
- Adding an ActiveReport to a Visual Studio project
- Connecting the report to a data source
- Creating a dataset to populate the parameter values
- Adding a report parameter
- Adding a dataset for the report
- Adding controls to the report to contain data
- Setting up a recursive hierarchy
- Using the Level function to display the hierarchy
- Viewing the report

> **Note:**
>
> - This walkthrough uses the **Store** table from the Reels database. The Reels.mdb file can be downloaded from GitHub: ..\Samples14\Data\Reels.mdb.
> - Although this walkthrough uses Page reports, you can also implement this using RDL reports.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout

Run-Time Layout



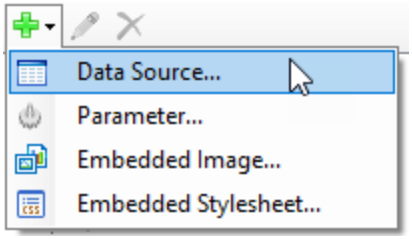
To add an ActiveReport to the Visual Studio project

1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 Page Report** and in the Name field, rename the file as rptRecursiveHierarchy.
4. Click the **Add** button to open a new fixed page report in the designer.

See Quick Start for information on adding different report layouts.

To connect the report to a data source

1. In the Report Explorer, right-click the Data Sources node and select the **Add Data Source** option or select **Data Source** from the Add button.



2. In the Report Data Source Dialog that appears, select the **General** page and in the Name field, enter a name like ReportData.
3. On this page, create a connection to the Reels database. See Connect to a Data Source for information on connecting to a data source.

To create a dataset to populate the parameter values

1. In the Report Explorer, right-click the data source node and select the **Add Data Set** option or select **Data Set** from the Add button.
2. In the DataSet Dialog that appears, select the **General** page and enter the name of the dataset as Stores. This name appears as a child node to the data source icon in the Report Explorer.
3. On the **Query** page of this dialog, in the **Query** field enter the following SQL query.

SQL Query
`SELECT StoreID FROM Store`

4. Click the **Validate DataSet** icon at the top right hand corner above the Query box to validate the query.
5. Click **OK** to close the dialog. Your data set and queried fields appear as nodes in the Report Explorer.

To add a report parameter

1. In the Report Explorer, right-click the data source node and select the **Parameters** node or select **Parameter** from the Add button.
2. In the Report - Parameters dialog that appears, set the following values:

   In the General tab

   - Name: StoreID
   - DataType: Integer
   - Text for prompting users for a value: Select a store number

   In the Available Values tab select **From query**

   - Dataset: Stores
   - Value: StoreID
   - Label: StoreID
3. Click **OK** to close the dialog and add the parameter under the Parameters node of the Report Explorer.

To add a dataset for the report

1. In the Report Explorer, right-click the data source node and select the **Add Data Set** option.
2. In the DataSet Dialog that appears, select the **General** page and enter the name of the dataset as **Employees**. This name appears as a child node to the data source icon in the Report Explorer.
3. On the **Parameters** page, add a parameter with the following properties.
   - **Parameter Name**: @StoreID
   - **Value**: =Parameters!StoreID.Value
4. On the **Query** page of this dialog, change the **Command Type** to StoredProcedure and enter the following stored procedure into the Query text box (the question mark denotes the parameter)

StoredProcedure Query
`EmployeesForStore ?`

   EmployeesForStore ?
5. Click the **Validate** icon to validate the query and to populate the Fields list.
6. Click **OK** to close the dialog. Your data set and queried fields appear as nodes in the Report Explorer.

To create a layout for the report

1. From the toolbox, drag a Table control onto the body of the report and go to the Properties window to set the following properties.

Property Name	Property Value
Location	0in, 0in
DataSetName	Employees
FixedSize (only for Page reports)	6.5in, 9in
2. Click inside the table to display the row and column handles along the left and top edges of the table.
3. Select the columns by clicking the grey column header above the column and change the **Width** property in the order as follows.

Column	Property Name
Second Column	Width: 1.5in
Third Column	Width: 1.05in

First Column	Width: 2.2in

4. Right-click the grey column header above the third column and select **Insert Column to the Right**.
5. Select the fourth column and change its **Width** property to **1.75in**.

> **Tip:** Making some columns narrower before adding columns or making other columns wider prevents your report width from changing.

6. In the Report Explorer from the **Employees** dataset, drag the following fields onto the detail row of the table.

Data Field	Column Name
Title	TableColumn1
LastName	TableColumn2
Supervisor	TableColumn3
Department	TableColumn4

7. Select the **LastName** field in the second column of the detail row and in the Properties window, set the **Value** property to **=Fields!LastName.Value & " " & Fields!EmployeeID.Value**. This will display the Employee ID number along with each employee's last name.
8. Select the static label in the second column of the header row of the table and in the properties window, set its **Value** property to **Last Name and ID**.
9. Select the **Supervisor** field in the third column of the detail row and in the Properties window, set its **TextAlign** property to **Center**.
10. Select the static label in the third column of the detail row and in the Properties window, set its **Value** property to **Supervisor ID**.
11. Select the header row by clicking the table handle to the left of the row and in the Properties window, set the following properties.

Property Name	Property Value
TextAlign	Center
FontWeight	Bold
BackgroundColor	DarkSlateBlue
Color	White

12. Select the detail row and in the Properties window, set the **BorderStyle** property to **Solid**.
13. Right-click the table handle to the left of the header row and select **Insert Row Above**.
14. While holding down the CTRL key, click each of the cells in the newly added top row.
15. Right-click inside the selected cells and select **Merge Cells** to create a cell that spans the table.
16. Go to the Properties window to set the following properties.

Property Name	Property Value
TextAlign	Center
FontSize	14pt
Value	="Store Number " & Parameters!StoreID.Value

17. Right-click the table handle to the left of the row and select **Table Footer** to remove the footer row from the table.

> **Note**: In case you are setting a recursive hierarchy on a Page report, set the **DataSetName** property in the Fixed Page Dialog to **Employees**.

To set up a recursive hierarchy

1. Right-click the table handle to the left of the Detail row and select **Edit Group** to open the **Table-Detail Grouping** dialog.
2. Under **Group on: Expression**, select **=Fields!EmployeeID.Value**.
3. Under **Parent group:**, select **=Fields!Supervisor.Value**.
4. Click **OK** to close the dialog.

To use the Level function to display the hierarchy

1. Select the cell in the first column of the Detail row that reads =Fields!Title.Value and in the Properties window, expand the **Padding** property.
2. In the **Padding > Left** property, enter the expression =2 + (Level() * 15) & "pt".

> **Note**: Adding 2 to the result ensures that a normal amount of padding is always used.

To use the Level function with themes in the BackgroundColor property of a textbox

1. Download Reels.rdlx-theme file from GitHub.
2. Copy the file **Reels.rdlx-theme** and paste it into the folder in which you saved your report.
3. In the Report Explorer, select **Report**.
4. In the Properties window in the **Theme** property, enter the theme file name: **Reels.rdlx-theme**.
5. Click the table handle to the left of the detail row to select the entire row.
6. In the Properties window, set the **BackgroundColor** property to **=Theme.Colors(Level() + 1, 4)**.

To view the report

- Click the preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

Single Layout Reports

A Page Report, allows you to create a layout at design time on a single page tab and apply it to the entire report. Designing a layout in this format gives you the advantage of creating a layout that appears exactly the same at design time and run time. This walkthrough illustrates how to create a report that contains a single layout and preview it.

This walkthrough is split into the following activities:

- Adding an ActiveReport to a Visual Studio project
- Connecting to a the data source
- Adding a dataset
- Creating a layout
- Viewing the report

> **Note:**
>
> - This walkthrough uses the **DVDStock** table from the Reels database. The Reels.mdb file can be downloaded from GitHub: ..\Samples14\Data\Reels.mdb.
> - Although this walkthrough uses Page reports, you can also implement this using RDL reports.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout

Notice that the run time report layout below is similar to the one you see at design time except for the data which you see at run time or when you preview the report.

Run-Time Layout

To add an ActiveReport to the Visual Studio project

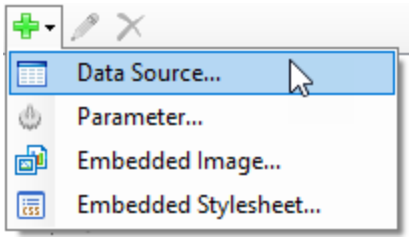
1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 Page Report** and in the Name field, rename the file as rptSingleLayout.
4. Click the **Add** button to open a new fixed page report in the designer.

See Quick Start for information on adding different report layouts.

To connect the report to a data source

1. In the Report Explorer, right-click the Data Sources node and select the **Add Data Source** option or select **Data**

Source from the Add button.



2. In the Report Data Source Dialog that appears, select the **General** page and in the Name field, enter a name like ReportData.
3. On this page, create a connection to the Reels database. See Connect to a Data Source for information on connecting to a data source.

To add a dataset

1. In the Report Explorer, right-click the data source node and select the **Add Data Set** option or select **Data Set** from the Add button.
2. In the DataSet Dialog that appears, select the **General** page and enter the name of the dataset as DVDList. This name appears as a child node to the data source icon in the Report Explorer.
3. On the **Query** page of this dialog, in the **Query** field enter the following SQL query.

SQL Query
`Select * from dvdstock`

4. Click the **Validate DataSet** icon at the top right hand corner above the Query box to validate the query.
5. Click **OK** to close the dialog. Your data set and queried fields appear as nodes in the Report Explorer.

To create a layout for the report

1. In the Visual Studio toolbox, go to the **ActiveReports 14 Page Report** tab and drag a TextBox control onto the design surface.
2. Select the TextBox control and go to the Properties window to set the following properties.

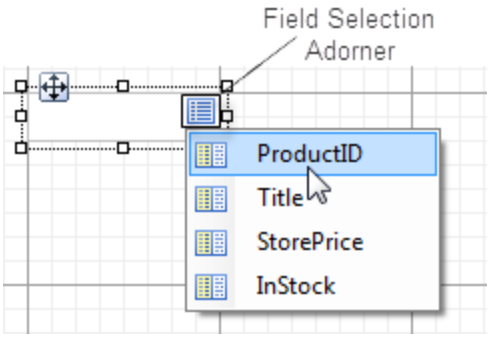
Property Name	Property Value
Location	0.25in, 0.5in
BackgroundColor	Khaki
Color	Maroon
Font	Normal, Arial, 16pt, Bold
Size	6in, 0.5in
TextAlign	Center
Value	DVD STOCK

3. From the Visual Studio toolbox, drag a Table data region and place it on the design surface.
4. Select the Table and go to the Properties window to set the following properties.

Property Name	Property Value
Location	0.25in, 1in
BackgroundColor	Khaki

FixedSize	6in, 7.5in
RepeatHeaderOnNewPage	True
Size	6in, 0.75in

5. In the Table data region, place your mouse over the cells of the table details row to display the field selection adorner.



6. Click the adorner to show a list of available fields from the DataSet and add the following fields to the cells of the table details row.

Cell	Field
Left Cell	Title
Middle Cell	InStock
Right Cell	StorePrice

This automatically places an expression in the details row and simultaneously places a static label in the header row of the same column.

> **Tip:** You can also directly drag fields from the Report Explorer onto the textbox cells of the Table data region.

7. Select the columns of the Table and set their **Width** property as follows:

Cell	Field
Left Cell	2.5in
Middle Cell	1.75in
Right Cell	1.75in

8. Select the following table rows and go to the Properties window to set their following properties.

Row	Properties
Table Header	BorderStyle: Solid Color: Maroon Font: Normal, Arial, 12pt, Bold TextAlign: Left
Table Details	BorderStyle: Solid Font: Normal, Arial, 10pt, Bold

	TextAlign: Left

To view the report

- Click the preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

Subreports in Page/RDL Reports

You can create a Page or an RDL report that hosts a subreport. This walkthrough illustrates how to create an RDL report using a subreport.

The walkthrough is split up into the following activities:

- Creating a report for the subreport
- Connecting the subreport to a data source
- Adding a dataset with a parameter to the subreport
- Adding a report parameter to the subreport
- Adding controls to display data on the subreport
- Creating the main report
- Connecting the main report to a data source
- Adding a dataset to the main report
- Adding controls to display data on the main report
- Viewing the report

> **Note:** This topic uses the **Employee, Sale and SaleDetails** tables in the Reels database. The Reels.mdb file can be downloaded from GitHub: ..\Samples14\Data\Reels.mdb.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout

Run-Time Layout

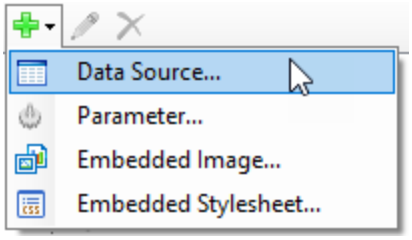
To add a report for the subreport

1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 RDL Report** and in the Name field, rename the file as **Sales.rdlx**.
4. Click the **Add** button to open a new RDL report in the designer.
5. In the Solution Explorer, select **Sales.rdlx** and set the **Build Action** property to **Content** and the **Copy to Output Directory** property to **Copy Always**.

See Quick Start for information on adding different report layouts.

To connect the subreport to a data source

1. In the Report Explorer, right-click the Data Sources node and select the **Add Data Source** option or select **Data Source** from the Add button.



2. In the Report Data Source Dialog that appears, select the **General** page and in the Name field, enter a name like **Reels**.
3. On this page, create a connection to the Reels database. See Connect to a Data Source for information on connecting to a data source.

To add a report parameter to the subreport

1. In the Report Explorer, right-click the Parameters node and select the **Add Parameter** option or select **Parameter** from the Add button.
2. Under **Name**, enter EmployeeID.
3. Under **Data type**, select Integer.
4. Click **OK** to close the dialog.

To add a dataset with a parameter to the subreport

When you add a query parameter using the syntax required by your database you must add a parameter to the Parameters page to ensure that the parameter value is passed to the query from the Report Parameters collection.

1. In the Report Explorer, right-click the data source node and select the **Add Data Set** option or select **Data Set** from the Add button.
2. In the DataSet Dialog that appears, select the **General** page and enter the name of the dataset as **EmployeeSales**. This name appears as a child node to the data source icon in the Report Explorer.
3. On the **Parameters** page under Parameter Name enter EmployeeID.
4. Under Value enter =Parameters!EmployeeID.Value
5. On the **Query** page of this dialog, in the **Query** field enter the following SQL query.

SQL Query
`SELECT * FROM EmployeeSales`

6. Click the **Validate DataSet** icon at the top right hand corner above the Query box to validate the query.

7. Click **OK** to close the dialog. Your data set and queried fields appear as nodes in the Report Explorer.

To add controls to display data on the subreport

1. From the toolbox, drag a **Table** data region onto the body of the report and go to the properties window to set the **DataSetName** property to **EmployeeSales**.
2. Click inside the table to display the column and row handles along the top and left sides of the table.
3. Right-click the handle above the rightmost column and select **Insert Column to the Right** to add another column.
4. Click the column handle at the top of each column in turn to select it, and in the property grid, set the **Width** property as indicated in the table.

Column	Width
First	1.5in
Second	1.5in
Third	1.2in
Fourth	1.55in

> **Tip:** In most cases it is easier to resize existing columns before adding new columns because this prevents the table from growing horizontally and pushing the report width beyond what will fit on paper.

5. Right-click the handle to the left of the table detail row and select **Insert Group** to open the Table-Groups dialog.
6. Under **Expression** select =Fields!EmployeeID.Value. This groups all details from each employee.
7. Change the **Name** to Employee and click **OK** to close the dialog. A grouping row is added to the table.

> **Note:** You cannot change the name of a table group until after you have set the expression.

8. Right-click the handle to the left of the table detail row and select **Edit Group** to access the **Table-Detail Grouping** dialog.
9. Under **Expression** select =Fields!SaleID.Value and click **OK** to close the dialog. This lists the total amount of each sale instead of listing each item sold within each SaleID.
10. Right-click the handle to the left of the grouping row and select **Insert Row Below**. We will use this new row for static labels that repeat at the top of each new group.
11. Right-click any handle to the left of the table and select **Table Header** to toggle off the table header.
12. Right-click any handle to the left of the table and select **Table Footer** to toggle off the table footer.
13. In the Report Explorer, select the Body node and go to the Properties window to set the **Size** property to 5.75in, 1in so that it fits inside the subreport control on the main report.

To add data fields to the Table data region

1. In the Report Explorer, from the EmployeeSales dataset, drag the following field onto the first group header row of the table.

Data Field	Column Name	Property Name
Name	TableColumn1	FontWeight: Bold

2. Use the Shift key and the mouse to select the first two cells in the first group header row, right-click and select **Merge Cells**. This allows the employee name to span two columns in the table.
3. Using the handle to the left of the first group header row, select the row and set the **BackgroundColor** property to **LightSteelBlue**.

> **Tip:** Even if you do not want to use colors in your finished report, it is often helpful to do so during the design of a report to make identification of the various sections easier for troubleshooting when you preview it.

4. Enter the following text into the cells in the second group header row of the table.

Data Field	Column Name	Property Name
Sale Date	TableColumn1	FontWeight: Bold TextAlign: Right
Sale Number	TableColumn2	FontWeight: Bold TextAlign: Right
Quantity	TableColumn3	FontWeight: Bold TextAlign: Right
Total	TableColumn4	FontWeight: Bold TextAlign: Right

5. Using the handle to the left of the second group header row, select the row and set the **BackgroundColor** property to LightGray.
6. In the Report Explorer, drag the following fields from the EmployeeSales dataset onto the detail row of the table.

Data Field	Column Name	Property Name
Sale Date	TableColumn1	Format: Short date
SaleID	TableColumn2	
Quantity	TableColumn3	
Total	TableColumn4	Format: Currency

7. In the detail row of the table, select the textbox with the **Quantity** data field and go to the Properties window to change the **Value** property to **=Sum(Fields!Quantity.Value)**. This adds the Sum aggregate to the expression for the field and shows a summary of the quantity field for each SalesID.
8. In the detail row of the table, select the textbox with the **Total** data field and go to the Properties window to change the **Value** property to **=Sum(Fields!Total.Value)**. This adds the Sum aggregate to the expression for the field and shows a summary of the total field for each SalesID.
9. In the Report Explorer, from the **EmployeeSales** dataset, drag the following fields onto the group footer row of the table. Notice that the value of fields dragged onto the group footer row automatically use the **Sum** aggregate function.

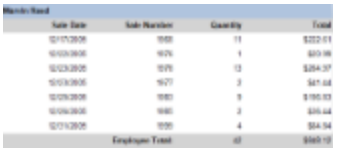
Data Field	Column Name	Property Name
Quantity	TableColumn3	Value: =Sum(Fields!Quantity.Value)
Total	TableColumn4	Format: Currency

		Value: =Sum(Fields!Total.Value)

10. Enter the following text into the indicated cell in the group footer row of the table.

Text	Column Name	Property Name
Employee Total:	TableColumn2	FontWeight: Bold TextAlign: Right

11. Using the handle to the left of the group footer row, select the row and in the **BackgroundColor** property select **LightGray**.
12. Go to the preview tab, enter **1035** for the Employee ID, and click the **View Report** button. You get a layout that looks similar to the following at design time and at run time.

Design-Time Layout	Run-Time Layout
	

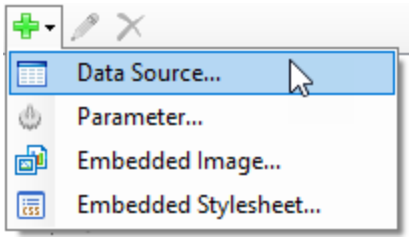
13. From the **File** menu, select **Save** and save this file. This report functions as the subreport you use in the main report.

To create the main report

1. From the Visual Studio **Project** menu, select **Add New Item**.
2. In the Add New Item dialog that appears, select **ActiveReports 14 RDL Report** and in the Name field, rename the file as **Employees.rdlx**.
3. Click the **Add** button to open a new fixed RDL report in the designer.
4. In the Report Explorer, select the Body node and go to the Properties window to set the **Size** property to 6.5in, 3.6in.

To connect the main report to a data source

1. In the Report Explorer, right-click the Data Sources node and select the **Add Data Source** option or select **Data Source** from the Add button.



2. In the Report Data Source Dialog that appears, select the **General** page and in the Name field, enter a name like **Reels**.
3. On this page, create a connection to the **Reels** database. See Connect to a Data Source for information on connecting to a data source.

To add a dataset to the main report

1. In the Report Explorer, right-click the data source node and select the **Add Data Set** option or select **Data Set** from the Add button.
2. In the DataSet Dialog that appears, select the **General** page and enter the name of the dataset as **EmployeeInfo**. This name appears as a child node to the data source icon in the Report Explorer.

3. On the **Query** page of this dialog, in the **Query** field enter the following SQL query.

SQL Query
`SELECT * FROM EmployeeInfo`

4. Click the **Validate** icon to validate the query and to populate the Fields list.
5. Click **OK** to close the dialog. Your data set and queried fields appear as nodes in the Report Explorer.

To add controls to display data on the main report

The following steps demonstrate how you can add controls and create the main report:

To add a static label to the top of the main report

From the toolbox, drag a TextBox control onto the body of the report and set the following properties:

Property Name	Property Value
Font	Normal, Arial, 14pt, Bold
Location	0in, 0in
Size	6.5in, 0.3in
TextAlign	Center
Value	Employee Report by City and Store

To add a List data region that repeats data for each city

1. Drag a List data region from the toolbox onto the body of the report and with the data region selected, go to the Properties Window to set the following properties:

Property Name	Property Value
BackgroundColor	Silver
DataSetName	EmployeeInfo
Location	0in, 0.5in
Size	6.5in, 3.1in

2. At the bottom of the Properties Window, select the **Property dialog** command. See Properties Window for further details on accessing commands.
3. In the List dialog that appears, select Detail Grouping.
4. Under Expression, select =Fields!City.Value
5. Click **OK** to close the dialog.
6. From the Report Explorer, drag the **City** field onto the List data region and set the following properties:

Property Name	Property Value
FontSize	12pt
Location	0in, 0in
Size	6.5in, 0.25in
TextAlign	Center

To nest a second List data region that repeats data for each store within the city

1. Drag a List data region from the toolbox onto the the first list and with the data region selected, go to the Properties Window to set the following properties:

Property Name	Property Value
BackgroundColor	Beige
DataSetName	EmployeeInfo
Location	0.125in, 0.3in
Size	6.25in, 2.7in

2. At the bottom of the Properties Window, select the **Property dialog** command. See Properties Window for further details on accessing commands.
3. In the List dialog that appears, select Detail Grouping.
4. Under Expression, select =Fields!StoreName.Value
5. Click **OK** to close the dialog.
6. From the Report Explorer, drag the **StoreName** field onto the list and set the following properties:

Property Name	Property Value
FontWeight	Bold
Location	0in, 0in
Size	2in, 0.25in

To nest a third List data region that repeats data for each employee in the store

1. Drag a List data region from the toolbox onto the second list and with the data region selected, go to the Properties Window to set the following properties:

Property Name	Property Value
BackgroundColor	White
DataSetName	EmployeeInfo
Location	0in, 0.25in
Size	6.125in, 2.125in

2. At the bottom of the Properties Window, select the **Property dialog** command. See Properties Window for further details on accessing commands.
3. In the List dialog, select Detail Grouping.
4. Under Expression, select =Fields!EmployeeID.Value
5. Click **OK** to close the dialog.
6. From the Report Explorer, drag the following fields onto the list and set the following properties:

Data Field	Property Name
Name	Location: 1.125in, 0in Size: 2.625in, 0.25in
Education	Location: 1.125in, 0.25in Size: 2.625in, 0.25in

DateOfBirth	Location: 5in, 0in Size: 0.875in, 0.25in Format: Short date
PhoneNumber	Location: 4.875in, 0.25in Size: 1in, 0.25in

7. From the toolbox, drag five text boxes onto the List and set the following properties:

TextBox Name	Value Property	Property Name
TextBox 1	Name:	Location: 0.125in, 0in Size: 0.625in, 0.25in FontWeight: Bold
TextBox 2	Education:	Location: 0.125in, 0.25in Size: 0.875in, 0.25in FontWeight: Bold
TextBox 3	Date of Birth:	Location: 3.875in, 0in Size: 1in, 0.25in FontWeight: Bold
TextBox 4	Phone:	Location: 3.875in, 0.25in Size: 0.875in, 0.25in FontWeight: Bold
TextBox 5	Sales Record	Location: 0.125in, 0.5in Size: 1in, 0.25in FontWeight: Bold

To add a Subreport control to the main report

1. From the toolbox, drag a Subreport control onto the third list and with the control selected, go to the Properties Window to set the following properties:

Property Name	Property Value
Location	0.125in, 0.75in
NoRows	No sales recorded for this employee during 2005.
ReportName	Sales (ensure that this report is saved in the same directory as the Sales report) **Note:** To view the report in the preview tab, you should specify the full path to the subreport.
Size	5.75in, 1.3in
Visibility: Hidden	True (hides the subreport initially)
Visibility:	Sales Record text box added in the previous

ToggleItem	procedure (puts a toggle image next to the text that shows the subreport when clicked)

2. At the bottom of the Properties Window, select the **Property dialog** command. See Properties Window for further details on accessing commands.
3. On the **Parameters** page of the Subreport dialog, set the **Parameter Name** to EmployeeID. This name must match the parameter in the subreport exactly.
4. Set the **Parameter Value** to =Fields!EmployeeID.Value.

   > **Note:** You can use the option of having the subreport automatically apply the same theme as the hosting report. This option is available on the General page of the Subreport Properties.

5. Click **OK** to close the dialog.

To view the report

- Click the preview tab to view the report.

OR

- See Windows Forms Viewer to display report in the Viewer at run time.

> **Note:** Click the + to the left of **Sales Record** to see the subreport.

Chart

This section contains the walkthroughs on creating different types of charts.

Column Charts
: This walkthrough demonstrates how to create a simple Column chart.

Funnel Charts
: This walkthrough demonstrates how to create a Funnel chart.

Gantt Charts
: This walkthrough demonstrates how to create a Gantt chart.

Column Charts (Classic Charts)
: This walkthrough demonstrates how to create a simple Column chart (classic).

Column Charts

You can create a page report with a chart using the ActiveReports Chart data region. This walkthrough illustrates how to create a simple report with a Column chart.

The walkthrough is split into the following activities:

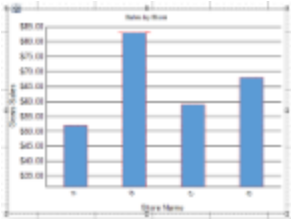
- Creating an ActiveReports project in Visual Studio
- Connecting the report to a data source
- Adding a dataset
- Adding a chart data region with data and grouping to the report
- Configuring the appearance of the chart
- Viewing the report

Note:

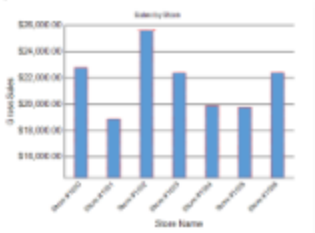
- This walkthrough uses the **StoreSummary** table from the Reels database. The Reels.mdb file can be downloaded from GitHub: ..\Samples14\Data\Reels.mdb.
- Although this walkthrough uses Page reports, you can also implement this using RDL reports.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout



Run-Time Layout



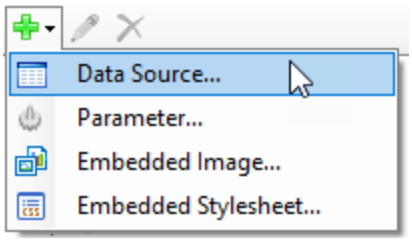
To create an ActiveReports project in Visual Studio

1. Create a new Visual Studio project.
2. In the **New Project** dialog that appears, select **ActiveReports 14 Page Report Application** and in the Name field, rename the file as **rptSalesByStore**.
3. Click **OK** to create a new **ActiveReports 14 Page Report Application**. By default a Page report is added to the project.

See Quick Start for information on adding different report layouts.

To connect the report to a data source

1. In the Report Explorer, right-click the Data Sources node and select the **Add Data Source** option or select **Data Source** from the Add button.



2. In the Report Data Source Dialog that appears, select the **General** page and in the Name field, enter a name like **ChartData**.
3. On this page, create a connection to the Reels database. See Connect to a Data Source for information on

connecting to a data source.

To add a dataset

1. In the Report Explorer, right-click the **ChartData** data source node and select the **Add Data Set** option or select **Data Set** from the Add button.
2. In the DataSet Dialog that appears, select the **General** page and enter the name of the dataset as **StoreSummaryData**. This name appears as a child node to the data source icon in the Report Explorer.
3. On the **Query** page of this dialog, in the **Query** field enter the following SQL query.

SQL Query
`SELECT * from StoreSummary`

4. Click the **Validate DataSet**  icon at the top right hand corner above the Query box to validate the query.
5. Click **OK** to close the dialog. Your data set and queried fields appear as child nodes to the data source in the Report Explorer.

To add chart data region with data and grouping to the report

1. From the toolbox, drag the Chart data region onto the design surface of the report .
2. In the **Select a Chart Type** wizard that appears, select the chart type **Column.**
3. Go to the Properties Window, click ellipses button next to **Plots** property to open the **PlotDesigner Collection Editor.** You see that Plot1 is already added.
4. Select Plot1 and click ellipses next to go to **Encodings>Values** property to open the **DataFields Collection Editor**.
5. Add a field and set its value to expression `=Fields!GrossSales.Value`.
6. Click **OK** to save and return to PlotDesigner Collection Editor.
7. Go to **Encodings>Category>Values** property and click ellipses next to Values property.
8. Add a field and set its value to expression `=Fields!StoreName.Value`.
9. Click OK to save and exit expression editor and then again to close the collection editor.
   You see that 'Store Name' is added to Category Fields and 'Gross Sales' is added to Data Fields.

To configure the appearance of the chart

1. Select Chart and set **Pallet** to Office.
2. Select 'Chart title' on the chart and set **Title** to 'Sales by Store'.
3. Select the chart plot and set following properties.

Property Name	Property Value
LineStyle>LineColor	Red
LineStyle>LineStyle	Dotted

4. Select X-axis and set **Labels>LabelsAngle** to -45 degrees.
5. Select X-axis and set Title>TitleStyle>Padding>Top to 8pt.
6. Select Y-axis and set Title>TitleStyle>Padding>Right to 4pt.

> Note: You may need to adjust:
> - Size of the chart so that all labels are displayed correctly. Select Chart and set **Size** property to a larger value.
> - Page size of the report so that the chart is fully rendered. Select Report and set **PageSize** property to a larger value.

To view the report

- Click the Preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

Composite Charts

You can create Page and RDL reports with composite charts using the ActiveReports Chart data region. A composite chart consists of two or more series plotted along the Y-axis, where each series can display a different chart type. In ActiveReports, a composite chart may have up to six Y-axis series. You can combine the following chart types:

- Column: Plain, Stacked, Percent Stacked
- Area: Plain, Stacked, Percent Stacked
- Line: Plain, Smooth

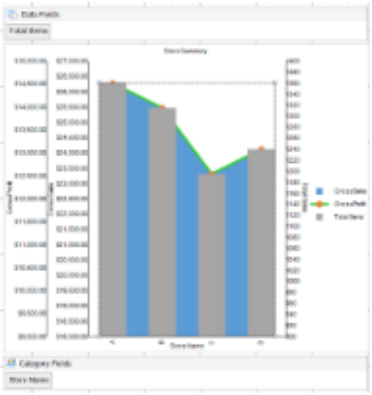
This walkthrough illustrates a step-by-step implementation for creating a composite chart with three Y-axis series. The walkthrough is split into the following activities:

- Creating an ActiveReports project in Visual Studio
- Connecting the report to a data source
- Adding a dataset
- Adding a chart data region and define plots
- Defining axes for plots
- Configuring the appearance of the chart
- Viewing the report

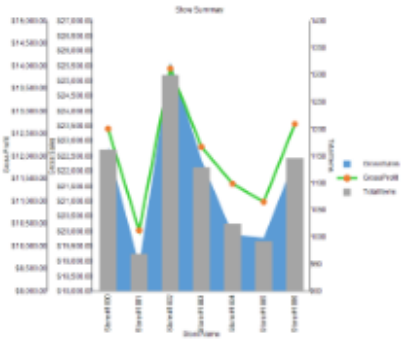
> **Note:**
>
> - This walkthrough uses the **StoreSummary** table from the Reels database. The Reels.mdb file can be downloaded from GitHub: ..\Samples14\Data\Reels.mdb.
> - Although this walkthrough uses Page reports, you can also implement this using RDL reports.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout



Run-Time Layout



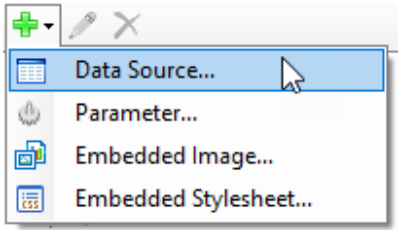
To create an ActiveReports project in Visual Studio

1. Create a new Visual Studio project.
2. In the **New Project** dialog that appears, select **ActiveReports 14 Page Report Application** and in the Name field, rename the file as **rptCompositeChart**.
3. Click **OK** to create a new **ActiveReports 14 Page Report Application**. By default a Page report is added to the project.

   See Quick Start for information on adding different report layouts.

To connect the report to a data source

1. In the Report Explorer, right-click the Data Sources node and select the **Add Data Source** option or select **Data Source** from the Add button.



2. In the Report Data Source Dialog that appears, select the **General** page and in the Name field, enter a name like **ChartData**.
3. On this page, create a connection to the Reels database. See Connect to a Data Source for information on connecting to a data source.

To add a dataset

1. In the Report Explorer, right-click the **ChartData** data source node and select the **Add Data Set** option or select **Data Set** from the Add button.
2. In the DataSet Dialog that appears, select the **General** page and enter the name of the dataset as **StoreSummaryData**.
3. On the **Query** page of this dialog, in the **Query** field enter the following SQL query.

SQL Query
`SELECT * from StoreSummary`

4. Click the **Validate DataSet** icon at the top right hand corner above the Query box to validate the query.
5. Click **OK** to close the dialog. Your data set and queried fields appear as child nodes to the data source in the Report Explorer.

To add the chart data region and define plots

1. From the toolbox, drag a Chart data region onto the design surface of the report.
2. In the **Select a Chart Type** wizard that appears, select the chart type as **Column.**

3. Go to the Properties Window, click ellipses button next to **Plots** property to open the **PlotDesigner - Collection Editor.** You see that default Plot1 is already added.
4. Add two new plots and set their **Name** as Plot2 and Plot3. Now you have three chart plots - Plot1, Plot2, and Plot3.
5. Set the properties of Plot1, Plot2, and Plot3 in the collection editor as follows.

Property Name	Property Values		
	Plot1	Plot2	Plot3
Encodings>Category>Values Bind field 'StoreName' to X-axis for each plot.	=Fields!StoreName.Value	=Fields!StoreName.Value	=Fields!StoreName.Value
Encodings>Values collection editor Bind fields 'GrossSales', 'GrossProfit', and 'TotalItems' to Y-axis for each plot, respectively.	=Fields!GrossSales.Value	=Fields!GrossProfit.Value	=Fields!TotalItems.Value
Encodings>Color>ShowValuesName Display Legends for each plot.	True	True	True
LineStyle>LineColor Display selected color as the color of line in respective plots.	Black	LimeGreen	Black
LineStyle>LineStyle Display selected style as the style of line in respective plots.	Solid	Solid	Solid
LineStyle>LineWidth Display selected width as the width of line (in pts) in respective plots.	1pt	3pt	1pt

6. Click OK to save and exit the collection editor.
7. Right click Plot 1 and select Plot Template > Area > Area.
8. Right click Plot 2 and select Plot Template > Line > Line.
9. Right click Plot 3 and select Plot Template to Column > Column.

Define axes for the plots

1. From Report Explorer, select Chart.
2. Go to the Properties Window and click ellipses next to **PlotArea > Axes** property to open the **AxisDesigner - Collection Editor.** You see that six axes are already added.
3. Remove Axis3 (AxisType X for Plot2) and Axis5 (AxisType X for Plot3). So now we have total four axes (one X axis - common for three plots and three Y axes - for three plots).
4. Set the properties of the four axes in the collection editor as follows.

Property Name	Property Values			
	Axis1	Axes2	Axes4	Axes6
Common>AxisType Specify axis type as X or Y.	X	Y	Y	Y
Common>Plots	Plot1, Plot2, Plot3	Plot1	Plot2	Plot3

Specify plots on respective axes.				
Title>Title Specify title to be displayed for each axis.	Store Name	Gross Sales	Gross Profit	Total Items
Labels>Format Specify format of the values displayed on each axis.	Default	c	c	d
Line>LineStyle>LineStyle Specify axis line styles.	Solid	Solid	Solid	Solid
Line>LineStyle>ShowLine Show axis lines on the plot.	True	True	True	True
Layout>Position Specify position of axes.	Near	Near	Near	Far
Major Grid > ShowMajorGrid Show or hide major grid lines.	True	False	False	False

5. Click OK to save and exit the collection editor.

To configure the appearance of the chart

1. Select Chart and set **Pallet** to Office.
2. Select 'Chart title' on the chart and set **Title** to 'Store Summary'.
3. Select X-axis and set **Labels>LabelsAngle** to 90 degrees.

   Let us set the maximum and minimum values to limit the values for each Y axis plots. This is based on the values in the data set.

4. Go to the Properties Window and click ellipses next to **PlotArea > Axes** property to open the **AxisDesigner - Collection Editor.** Note that axes get renamed once you reopen the AxisDesigner - Collection Editor.

Property Name	Property Values		
	Axes2 (Y axis for Plot1)	Axes3 (Y axis for Plot2)	Axes4 (Y axis for Plot3)
Scale>Max	27000	15000	1400
Scale>Min	18000	9000	900

Note: You may need to adjust:
- Size of the chart so that all labels are displayed correctly. Select Chart and set **Size** property to a larger value.
- Page size of the report so that the chart is fully rendered. Select Report and set **PageSize** property to a larger value.

To view the report

- Click the Preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

Funnel Charts

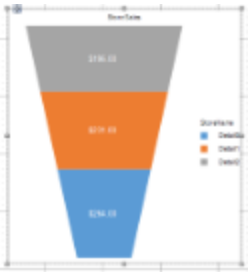
The Funnel charts enable users to visualize stages in a linear process. This walkthrough illustrates a step-by-step implementation for creating a funnel chart. The walkthrough is split into the following activities:

- Creating an ActiveReports project in Visual Studio
- Connecting the report to a data source
- Adding a dataset
- Adding a chart data region and define plots
- Configuring the appearance of the chart
- Viewing the report

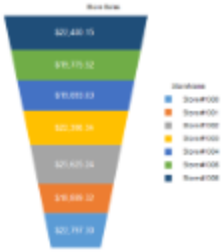
> **Note:**
>
> - This walkthrough uses the **StoreSales** table from the Reels database. The Reels.mdb file can be downloaded from GitHub: ..\Samples14\Data\Reels.mdb.
> - Although this walkthrough uses Page reports, you can also implement this using RDL reports.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout



Run-Time Layout



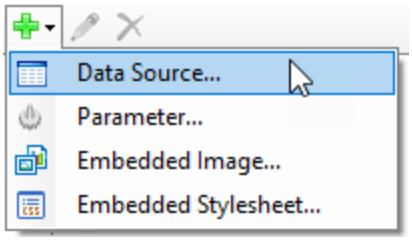
To create an ActiveReports project in Visual Studio

1. Create a new Visual Studio project.
2. In the **New Project** dialog that appears, select **ActiveReports 14 Page Report Application** and in the Name field, rename the file as **rptFunnelChart**.
3. Click **OK** to create a new **ActiveReports 14 Page Report Application**. By default a Page report is added to the project.

See Quick Start for information on adding different report layouts.

To connect the report to a data source

1. In the Report Explorer, right-click the Data Sources node and select the **Add Data Source** option or select **Data Source** from the Add button.



2. In the Report Data Source Dialog that appears, select the **General** page and in the Name field, enter a name like **ChartData**.
3. On this page, create a connection to the Reels database. See Connect to a Data Source for information on connecting to a data source.

To add a dataset

1. In the Report Explorer, right-click the **ChartData** data source node and select the **Add Data Set** option or select **Data Set** from the Add button.
2. In the DataSet Dialog that appears, select the **General** page and enter the name of the dataset as **StoreSalesData**.
3. On the **Query** page of this dialog, in the **Query** field enter the following SQL query.

SQL Query
`SELECT * from StoreSales`

4. Click the **Validate DataSet** icon at the top right hand corner above the Query box to validate the query.
5. Click **OK** to close the dialog. Your data set and queried fields appear as child nodes to the data source in the Report Explorer.

To add a chart data region and define plots

1. From the toolbox, drag a Chart data region onto the design surface of the report.
2. In the **Select a Chart Type** wizard that appears and select the chart type as **Column.**
3. Right-click on the chart and go to **Plot Template > Miscellaneous** and select **Funnel**.
4. Select the chart plot and go to Properties window.
5. Go to **Encodings > Values property** and click ellipses next to **Values** property to open **ValueAggregateEncodingDesigner Collection Editor**.
6. Add a new Value field to the collection editor.
7. Open its Value collection editor and add a value field and set its expression to `=Fields!TotalSales.Value`.
8. Click OK to save and exit the collection editor. You see that 'Total Sales' field is added under the Data Fields on chart.
9. Select the chart plot and go to Properties window.
10. Go to **Encodings > Details** property and click ellipses next to **Values** property to open **DetailEncodingDesigner Collection Editor**.
11. Add a new Detail field to the collection editor.
12. Open its Values collection editor and add a value field and set its expression to `=Fields!StoreName.Value`.
13. Click OK to save and exit expression editor and then again to close the collection editor. You see that 'Store Name' is added as Details encoding on the chart.
14. Select 'Store Name' encoding and set following properties:

Property Name	Property Values
Group	Stack
SortDirection	Ascending
SortingField	=Fields!StoreName.Value

Let us add legend to the chart.

15. Select the chart plot and go to **Encodings > Color > Values** property.
16. Click ellipses next to Values property to open the expressions editor.
17. Add a value field and set its expression to `=Fields!StoreName.Value`

To configure the appearance of the chart

1. Select Chart and set **Pallet** to Office.
2. Select 'Chart title' on the chart and set **Title** to 'Store Sales'.
3. Select Category axis and set **Lines>ShowLine** and **Labels>ShowLabels** to False.
4. Select Value axis and set **ShowLine** and **ShowLabels** to False.
5. Select the chart plot and set the label settings for the Funnel chart as:

Property Name	Property Values
Labels>Color	White
Labels>Template Specify the template to be displayed for the label.	{valueField.value:c}
Labels>TextPosition Specify text position of the label.	Center

6. Select Legends on chart and set Title to 'Store Name'.

To view the report

- Click the Preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

Gantt Charts

The Gantt charts are used to represent the schedule of sequence of tasks in a project. To plot Gantt charts, you need to have the start and end times of a task.
The walkthrough is split into the following activities:

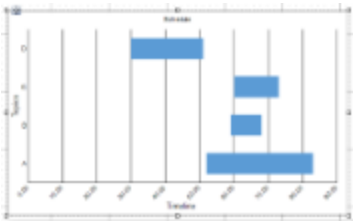
- Creating an ActiveReports project in Visual Studio
- Connecting the report to a data source
- Adding a dataset
- Adding a chart data region and define plots
- Configuring the appearance of the chart
- Viewing the report

Note:

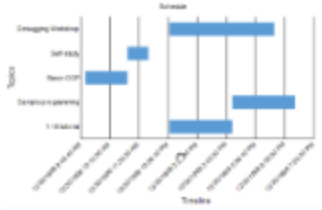
- This walkthrough uses the **ScheduleOfTime** table from the **Schedule** database. The Schedule.mdb can be downloaded from GitHub: ..\Samples14\Data\Reels.mdb.
- Although this walkthrough uses Page reports, you can also implement this using RDL reports.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout



Run-Time Layout



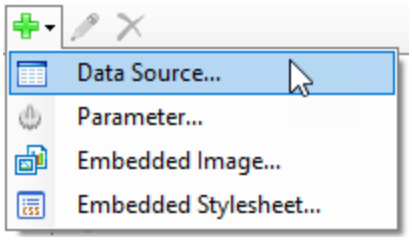
To create an ActiveReports project in Visual Studio

1. Create a new Visual Studio project.
2. In the **New Project** dialog that appears, select **ActiveReports 14 Page Report Application** and in the Name field, rename the file as **rptGanttChart**.
3. Click **OK** to create a new **ActiveReports 14 Page Report Application**. By default a Page report is added to the project.

   See Quick Start for information on adding different report layouts.

To connect the report to a data source

1. In the Report Explorer, right-click the Data Sources node and select the **Add Data Source** option or select **Data Source** from the Add button.



2. In the Report Data Source Dialog that appears, select the **General** page and in the Name field, enter a name like **ChartData**.
3. On this page, create a connection to the Schedule database. See Connect to a Data Source for information on

connecting to a data source.

To add a dataset

1. In the Report Explorer, right-click the **ChartData** data source node and select the **Add Data Set** option or select **Data Set** from the Add button.
2. In the DataSet Dialog that appears, select the **General** page and enter the name of the dataset as **ScheduleData**.
3. On the **Query** page of this dialog, in the **Query** field enter the following SQL query.

SQL Query
`SELECT Top 5 * from ScheduleOfTime`

4. Click the **Validate DataSet** icon  at the top right hand corner above the Query box to validate the query.
5. To obtain end time of the tasks, you need to add a field that calculates end times. Go to **Fields** tab and add a calculated field **EndTime** with expression `=DateAdd(DateInterval.Hour, [Hours], [StartTime])`.
6. Click **OK** to close the dialog. Your data set and queried fields appear as child nodes to the data source in the Report Explorer.

To add a chart data region and define plots

1. From the toolbox, drag a Chart data region onto the design surface of the report.
2. In the **Select a Chart Type** wizard that appears and select the chart type as **Column.**
3. Right-click on the chart and go to **Plot Template > Miscellaneous** and select **Gantt**.

   Note that the chart changes to horizontal bar chart, where horizontal axis (AxisType Y) represents time duration that a task needs to finish and each bar (plotted along Vertical axis (AxisType X)) represents a task.
4. Select the chart control so that the chart adorners are displayed.
5. Right-click the **Data Fields** adorner and click Add. You see that a field is added.
6. Select the field and click the ellipses next to **Subfields** property to open **Complex Fields Collection Editor**.
7. Set lower value to `=Fields!StartTime.Value` and upper value to `=Fields!EndTime.Value` to show Value axis.
8. Click OK to save and exit the collection editor.
9. Select the chart control so that the chart adorners are displayed.
10. Right-click the **Category Fields** adorner and click Add. You see that category 'Category0' is added.
11. Select 'Category0' and set its Expression to `=Fields!Labels.Value`.

To configure the appearance of the chart

1. Select Chart and set **Pallet** to Office.
2. Select 'Chart title' on the chart and set **Title** to 'Schedule'.
3. Select Category axis (AxisType X) and set **Title** to 'Topics'.
4. Select Value axis (AxisType Y) and set **Title** to 'Timeline'.
5. Select the Timeline axis and set the label settings as:

Property Name	Property Values
Labels>LabelAngle	-45
Title>TitleStyle>Font>FontSize Specify the template to be displayed for the label.	10pt

6. Select the Topics axis and set Title>TitleStyle>Font>FontSize to 10 pt.

To view the report

- Click the Preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

Column Charts (Classic Charts)

You can create a page report with a chart using the ActiveReports Chart data region. This walkthrough illustrates how to create a report with a chart.

The walkthrough is split into the following activities:

- Creating an ActiveReports project in Visual Studio
- Connecting the report to a data source
- Adding a dataset
- Adding a chart data region with data and grouping to the report
- Configuring the appearance of the chart
- Viewing the report

> **Note:**
>
> - This walkthrough uses the **Sales** table from the Reels database. The Reels.mdb file can be downloaded from GitHub: ..\Samples14\Data\Reels.mdb.
> - Although this walkthrough uses Page reports, you can also implement this using RDL reports.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout

Run-Time Layout

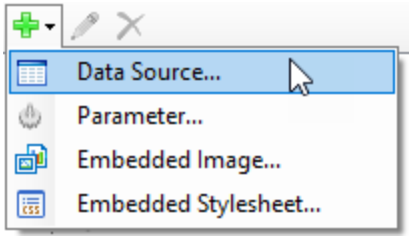
To create an ActiveReports project in Visual Studio

1. Create a new Visual Studio project.
2. In the **New Project** dialog that appears, select **ActiveReports 12 Page Report Application** and in the Name field, rename the file as **ProfitsByGenre**.
3. Click **OK** to create a new **ActiveReports 12 Page Report Application**. By default a Page report is added to the project.

See Quick Start for information on adding different report layouts.

To connect the report to a data source

1. In the Report Explorer, right-click the Data Sources node and select the **Add Data Source** option or select **Data Source** from the Add button.



2. In the Report Data Source Dialog that appears, select the **General** page and in the Name field, enter a name like ReportData.
3. On this page, create a connection to the Reels database. See Connect to a Data Source for information on connecting to a data source.

To add a dataset

1. In the Report Explorer, right-click the **ReportData** data source node and select the **Add Data Set** option or select **Data Set** from the Add button.
2. In the DataSet Dialog that appears, select the **General** page and enter the name of the dataset as **ProfitsByGenre**. This name appears as a child node to the data source icon in the Report Explorer.
3. On the **Query** page of this dialog, in the **Query** field enter the following SQL query.

SQL Query

```
SELECT Profit, SalesID, SaleDate, GenreName FROM SalesByGenre WHERE (SalesID < 1090) AND (GenreName = "Comedy" OR GenreName = "Drama" OR GenreName = "Adventure") ORDER BY SalesID
```

4. Click the **Validate DataSet** icon at the top right hand corner above the Query box to validate the query.
5. On the **Fields** page, add a new field by clicking the Add (+) button above the fields list.

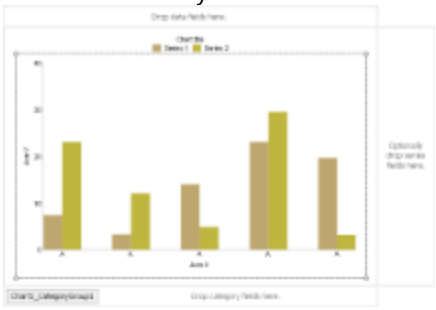
6. Set the **Name** to Month and the **Value** of the new field to `=Fields!SaleDate.Value.Month`
   This parses out the month from the date field.
7. Click **OK** to close the dialog. Your data set and queried fields appear as nodes in the Report Explorer.

To add chart data region with data and grouping to the report

1. From the toolbox, drag the Chart data region onto the design surface of the report .
2. In the wizard **Select a Chart Type** that appears, select the chart type **Column** and go to the Properties Window to set the following properties:

Property Name	Property Value
Location	0in, 0in
Size	6.5in, 4.5in

3. Double-click inside the chart area to display the UI along the top, right, and bottom of the chart to drop fields in.
4. From the Report Explorer, drag the **Month** field into the area below the chart labeled "Drop category fields here." This automatically binds the Month field to the X axis.



5. Right-click the month category group and select **Edit** to open the **Chart Data - Category Groups** dialog.
6. In the dialog that appears on the **General** page, go to the **Label** field and enter the following expression: `=MonthName(Fields!Month.Value)` and click **OK** to close the dialog.
7. From the ReportExplorer, drag the **Profit** field into the area above the chart labeled "Drop data fields here." This plots the profits to the Y axis of the chart.
8. In the Report Explorer, drag the **GenreName** field into the area to the right of the chart labeled "Drop series fields here." This sets the series to be charted in the chart area.
9. Right-click the genre series group and select **Edit** to open the **Chart Data - Series Groups** dialog.
10. In the dialog that appears on the **General** page, go to the **Label** field and enter the following expression: `=Fields!GenreName.Value`. This shows genre names in the legend.
11. Click **OK** to close the dialog.
12. Select the chart and at the bottom of the Properties Window, select the **Chart Data** command. See Properties Window for further details on accessing commands.
13. In the **Chart Data** dialog that appears, under **Series Values** go to the **Value** field and make sure it is set to `=Sum(Fields!Profit.Value)`.

To configure the appearance of the chart

1. Select the chart and at the bottom of the Properties Window, select the **Chart appearance** command.
2. In the **Chart Appearance** dialog that appears set the following:

   Title Page

   On the **Title** page enter Profits by Genre in the **Chart title** text box and change the **Size** property to 14pt.

   Palette Page

On the **Palette** page and select **Light** in the drop down list. This is a good color choice because it differentiates between genres without highlighting one of them unintentionally.

Plot Area Page

On the **Plot Area** page set the **Background Fill Color > Fill Color** to Silver.

3. Click **OK** to close the dialog.
4. Select the Chart Legends and at the bottom of the Properties Window, select the **Property dialog** command.
5. In the **Chart Legend - General** dialog that appears, uncheck the **Use smart settings** option.
6. Click **OK** to close the dialog.
7. Select the X-Axis label in the chart and at the bottom of the Properties Window, select the **Property dialog** command.
8. In the **Chart X-Axis - Title** dialog that appears set the following properties:

Title Page

On the **Title** page, in the **X-Axis Title** field remove the default "Axis X" text.

Labels Page

On the **Labels** page, set the **Size** property to 10pt.

9. Click **OK** to close the dialog.
10. Select the chart and at the bottom of the Properties Window, select the **Chart Y-axis** command.
11. In the **Chart Y-Axis - Title** dialog that appears set the following properties:

Title Page

On the **Title** page, in the **Y-Axis Title** field remove the default "Axis Y" text if available.

Labels Page

On the **Labels** page and in the **Format code** field, select Currency in the Format drop-down list. Set the **Size** property to 10pt.

Scale Page

On the **Scale** page and in the **Minimum** field enter 0 (zero). This sets the currency labels to start at zero on the Y axis.

12. Click **OK** to close the dialog.

To view the report

- Click the Preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

Composite Charts (Classic Charts)

You can create Page and RDL reports with composite charts using the ActiveReports Chart data region. A composite chart consists of two or more series plotted along the Y-axis, where each series can display a different chart type. In ActiveReports, a composite chart may have up to six Y-axis series. You can combine the following chart types:

- Column: Plain, Stacked, Percent Stacked
- Area: Plain, Stacked, Percent Stacked
- Line: Plain, Smooth

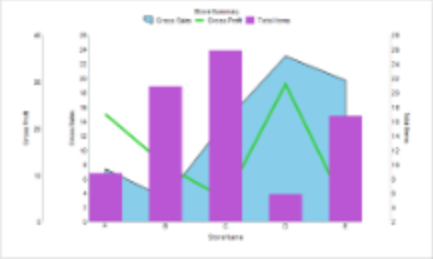
This walkthrough illustrates a step-by-step implementation for creating a composite chart with three Y-axis series. The walkthrough is split into the following activities:

- Creating an ActiveReports project in Visual Studio
- Connecting the report to a data source
- Adding a dataset
- Adding a chart data region with data to the report
- Configuring the appearance of the chart
- Viewing the report

> **Note:**
>
> - This walkthrough uses the **StoreSummary** table from the Reels database. The Reels.mdb file can be downloaded from GitHub: ..\Samples14\Data\Reels.mdb.
> - Although this walkthrough uses Page reports, you can also implement this using RDL reports.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout



Run-Time Layout

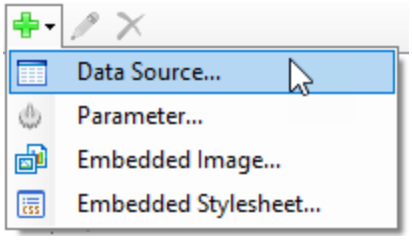
To create an ActiveReports project in Visual Studio

1. Create a new Visual Studio project.
2. In the **New Project** dialog that appears, select **ActiveReports 12 Page Report Application** and in the Name field, rename the file as **rptCompositeChart**.
3. Click **OK** to create a new **ActiveReports 12 Page Report Application**. By default a Page report is added to the project.

See Quick Start for information on adding different report layouts.

To connect the report to a data source

1. In the Report Explorer, right-click the Data Sources node and select the **Add Data Source** option or select **Data Source** from the Add button.



2. In the Report Data Source Dialog that appears, select the **General** page and in the Name field, enter a name like **ChartData**.
3. On this page, create a connection to the Reels database. See Connect to a Data Source for information on connecting to a data source.

To add a dataset

1. In the Report Explorer, right-click the **ChartData** data source node and select the **Add Data Set** option or select **Data Set** from the Add button.
2. In the DataSet Dialog that appears, select the **General** page and enter the name of the dataset as **StoreSummaryData**.
3. On the **Query** page of this dialog, in the **Query** field enter the following SQL query.

SQL Query
`SELECT * from StoreSummary`

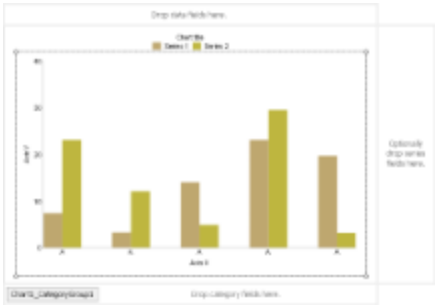
4. Click the **Validate DataSet** icon at the top right hand corner above the Query box to validate the query.
5. Click **OK** to close the dialog. Your data set and queried fields appear as child nodes to the data source in the Report Explorer.

To add the chart data region with data to the report

1. In the Report Explorer, select **Report** and in the Properties Window, set the **PaperOrientation** property to **Landscape**.
2. From the toolbox, drag a Chart data region onto the design surface of the report.
3. In the **Select a Chart Type** wizard that appears, select the chart type as **Column** and go to the Properties Window to set the following properties.

Property Name	Property Value
Location	0in, 0in
Size	7.5in, 4.5in

4. Double-click inside the chart area to display the chart panes along the top, right, and bottom of the chart to drop fields in.
5. From the Report Explorer, drag the **StoreName** field into the area below the chart labeled "Drop category fields here." This automatically binds the StoreName field to the X-axis.



6. Select the chart and at the bottom of Properties Window, select the **Chart Y-Axis** command.
7. In the **Y-axis** dialog that appears, under the **Title** tab, set the **Y-Axis title** property to **Gross Sales**.
8. Click **Add** to add the second Y-Axis. Set the **Y-Axis title** property to **Gross Profit** and the **Margin** property to **0.8in**.
9. Click **Add** to add the third Y-Axis. Set the **Y-Axis title** property to **Total Items** and the **Position** property to **Right**.
10. Click **OK** to close the dialog.
11. Select the chart and from the bottom of the Properties Window, select the **Chart data** command.
12. Select the **Series Values** page. Click **Add** to add the first Y-axis series and set the properties as follows.

Property Name	Property Value
Series label	Gross Sales
Value	=[GrossSales]
Chart Type	Area Plain
Y-Axis	Y1

13. Click **Add** to add the second Y-axis series and set the properties as follows.

Property Name	Property Value
Series label	Gross Profit
Value	=[GrossProfit]
Chart Type	Line Plain
Y-Axis	Y2

14. Click **Add** to add the third Y-axis series and set the properties as follows.

Property Name	Property Value
Series label	Total Items
Value	=[TotalItems]
Chart Type	Column Plain
Y-Axis	Y3

15. Click **OK** to close the dialog.

To configure the appearance of the chart

1. Select the chart and at the bottom of Properties Window, select the **Chart appearance** command.
2. In the **Chart Appearance** dialog that appears, on the **Title** page enter **Store Summary** in the **Chart title** field.
3. In the **Palette** page, open the Palette drop-down, and select **Pastel**.
4. Click **OK** to close the dialog.
5. Select the chart and at the bottom of the Properties Window, select the **Chart X-Axis** command.
6. In the **Chart X-Axis - Title** dialog that appears set the properties as follows.

Property Name	Property Value
X-Axis title	Store Name
Size	6pt

7. Click **OK** to close the dialog.
8. Select the chart and at the bottom of the Properties Window, select the **Chart Y-Axis** command.
9. Select **Y1** from the list of Y-axes and under the **Labels** tab, select the **Show y-axis labels** check box. Also, select Currency from the **Format code** drop-down.
10. Repeat the above step for **Y2**.
11. Select **Y3** from the list of Y-axes and under the **Labels** tab, select the **Show y-axis labels** check box.
12. Click **OK** to close the dialog.

To view the report

- Click the Preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

Map

This section contains the following walkthrough that fall under the Map category.

Reports with Map
: This walkthrough demonstrates how to create a report with a map.

Reports with Map

You can create a page report that contains a map using the ActiveReports Map control. The Map data region shows your data on a geographical background. This walkthrough illustrates how to create a report that uses a Map to display data.

The walkthrough is split into the following activities:

- Adding an ActiveReport to a Visual Studio project
- Connecting the report to a data source
- Adding a dataset
- Adding Map control to the report and configuring its data
- Configuring appearance of the Map
- Viewing the report

> **Note:**

- This walkthrough use tables from the Reels database. The Reels.mdb file can be downloaded from GitHub: ..\Samples14\Data\Reels.mdb.
- Although this walkthrough uses Page reports, you can also implement this using RDL reports.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout

Run-Time Layout

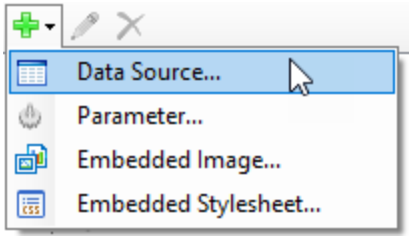
To add an ActiveReport to the Visual Studio project

1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 Page Report** and in the Name field, rename the file to **CustomersPopulation**.
4. Click the **Add** button to open a new fixed page report in the designer.

See Quick Start for information on adding different report layouts.

To connect the report to a data source

1. In the Report Explorer, right-click the Data Sources node and select the **Add Data Source** option or select **Data Source** from the Add button.



2. In the Report Data Source Dialog that appears, select the **General** page and in the Name field, enter a name like ReportData.
3. On this page, create a connection to the Reels database. See Connect to a Data Source for information on connecting to a data source.

To add a dataset

1. In the Report Explorer, right-click the data source node and select the **Add Data Set** option or select **Data Set** from the Add button.
2. In the DataSet Dialog that appears, select the **General** page and enter the name of the dataset as **Customers**. This name appears as a child node to the data source icon in the Report Explorer.
3. On the **Query** page of this dialog, in the **Query** field enter the following SQL query.

SQL Query

```
SELECT Address.Region, Customer.CustomerID FROM (Address INNER JOIN Person ON Address.[AddressID] = Person.[AddressID]) INNER JOIN Customer ON Person.[PersonID] = Customer.[PersonID];
```

4. Click the **Validate DataSet** icon at the top right hand corner above the Query box to validate the query.

5. Click **OK** to close the dialog. Your data set and queried fields appear as nodes in the Report Explorer.

To add a Map data region and configure its data

1. From the Visual Studio toolbox, drag and drop a Map control onto the design surface.
2. In the **Select a Map Template** wizard that appears, select the **USA Map** template.
3. Click the Map until the map panes appear.
4. In the layers pane, right click **PolygonLayer1** and select **Layer Data** to open **Map Layer Data Properties** dialog.
5. In the **Map Layer Data Properties** dialog that appears, go to the **Analytical data** page.
6. Select **Customers** from the **Dataset** property combo box and then click the **Add** (+) button located next to the **Match** label. This creates an empty match item and enables its **Spatial** and **Analytical** fields editor.

> **Note**: It is necessary to set match fields if you want to use a spatial data field from analytical data, or if you want to visualize analytical data on the map layer. Match fields enable the report processor to build a relationship between the analytical data and the spatial data.

7. In the **Spatial field** property, select `STATE_ABBR` from the combo box; and similarly, select `=Fields!Region.Value` in the **Analytical field** property. This builds the match field expression and relates the analytical data to map elements on a polygon layer.
8. Click **OK** to close the dialog.

To configure appearance of the Map

1. In the layers pane, right click **PolygonLayer1** and select **Edit** to open **Map Polygon Layer** dialog.

2. In the **General** page of the dialog, select `#STATE_NAME` from the **Label Text** combo box to display as label inside polygons at run time.
3. Go to the **Color Rule** page of the dialog, and select **Visualize data by using color palette** option. This activates the tabs below.
4. On the General tab, enter the following expression `=Count([CustomerID])` in the **Data field** property and set **Palette** property to SemiTransparent.
5. On the Distribution tab, set the **Method** property to EqualInterval.
6. On the Legends tab, click to select **Show in Legend**.
7. In Legend Name, enter **Legend**. This name relates to the default legend that appears in the Legend collection.
8. Click **OK** to close the dialog.
9. On the design surface, click on the Map control to select it and go to the Properties Window to set the following properties:

   Properties

Property Name	Property Value
BackgroundColor	White
BackgroundGradientEndColor	White
BorderStyle	Solid
ColorScale > Hidden	True
DistanceScale > Hidden	True
Location	0in, 0.625in
Size	6.5in, 4.75in
ViewPort > BackgroundColor	LightSteelBlue
ViewPort > BackgroundGradientEndColor	White
ViewPort > BorderStyle	None
ViewPort > CoordinateSystem	Planar
ViewPort > Margin > Right	20pt
ViewPort > Meridians > Hidden	True
ViewPort > Parallels > Hidden	True
ViewPort > View > Zoom	115

10. With the Map control selected, go to the Properties window, click the **Legends (Collection)** property and then click the ellipsis button that appears.
11. In the **LegendDesigner Collection Editor** that appears, under the **Members** list, select the existing legend and set the following properties:

    Properties

Property Name	Property Value
BackgroundColor	LightSteelBlue

BackgroundGradientEndColor	White
Location > DockOutsideViewport	False
Location > DockPosition	RightBottom
Title > (Caption)	Number of Customers
Title > Font	Normal, Arial, 10pt, Bold

12. Click **OK** to close the dialog.
13. With the Map control selected, go to the Properties window, click the **Titles (Collection)** property and then click the ellipsis button that appears.
14. In the **MapTitleDesigner Collection Editor** that appears, with **Title** selected in the Members list set the following properties:

    Properties

Property Name	Property Value
(Text)	Customers Population
Color	Black

15. Click **OK** to close the dialog.

To view the report

- Click the preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

Tablix

This section contains the following walkthrough that fall under the Tablix category.

Grouping in Tablix
: This walkthrough demonstrates how to use grouping in Tablix.

Cell Merging in a Row Group Area in Tablix
: This walkthrough demonstrates cells with duplicate values in a row group area are merged in Tablix.

Grouping in Tablix

This walkthrough illustrates a step-by-step implementation for creating a report which uses the Tablix data region to display regional product sales.

The walkthrough is split into the following activities:

- **Creating an ActiveReports project in Visual Studio**
- **Connecting the report to a data source**
- **Adding a dataset**
- **Creating a layout for the report**

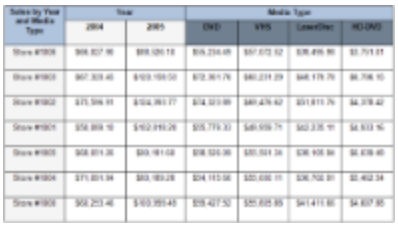
- **Enhancing the appearance of the report**
- **Viewing the report**

> **Note:**
>
> - This walkthrough uses the Reels database. The Reels.mdb file can be downloaded from GitHub: ..\Samples14\Data\Reels.mdb.
> - Although this walkthrough uses RDL reports, you can also implement this using page reports.

When you complete this walkthrough, you will have a layout that looks similar to the following at design time and at run time.

Design-Time Layout

Run-TimeLayout



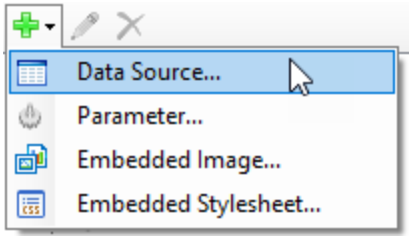
Creating an ActiveReports project in Visual Studio

1. Create a new Visual Studio project.
2. In the **New Project** dialog that appears, select **ActiveReports 14 RDL Report Application** and in the Name field, name the file rptTablix.
3. Click **OK** to create a new **ActiveReports 14 RDL Report Application**. By default an RDL report is added to the project.

   See Quick Start for information on adding different report layouts.

Connecting the report to a data source

1. In the Report Explorer, right-click the **Data Sources** node and select the **Add Data Source** option or select **Data Source** from the Add button.

2. In the Report Data Source Dialog that appears, select the **General** page and in the Name field, enter a name like ReportData.

3. On this page, create a connection to the Reels database. See Connect to a Data Source for information on connecting to a data source.

Adding a dataset

1. In the Report Explorer, right-click the **ReportData** node (the name of data source added) and select the **Add Data Set** option or select **Data Set** from the Add button.
2. In the DataSet Dialog that appears, select the **General** page and name the dataset **SalesData**. This name appears as a child node of the data source icon in the Report Explorer.
3. On the **Query** page of this dialog, in the Query field enter the following SQL query.

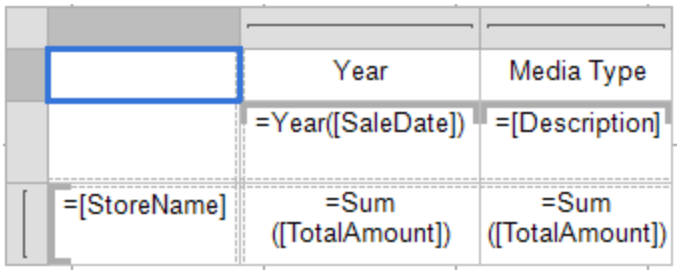
SQL Query

```
SELECT Sale.SaleDate, Sale.TotalAmount, MediaType.Description, MediaType.MediaID,
Sale.SalesID,
Sale.Store, Store.StoreName FROM Store INNER JOIN
(Sale INNER JOIN (MediaType INNER JOIN (MovieProduct INNER JOIN SaleDetails
ON MovieProduct.ProductID = SaleDetails.ProductID) ON MediaType.MediaID =
MovieProduct.MediaType) ON Sale.SalesID = SaleDetails.SaleID)
ON Store.StoreID = Sale.Store;
```

4. Click the **Validate DataSet** icon  at the top right hand corner above the Query box to validate the query.
5. Click **OK** to close the dialog. Your data set and queried fields appear as nodes in the Report Explorer.

Creating a layout for the report

1. From the toolbox, drag a Tablix data region onto the designer surface of the report.
2. Hover over TextBox2 to reveal the field selection adorner, click it to display a list of available fields, and select the **SaleDate** field. This is a column group cell, so selecting a field in it automatically adds a column group.
3. With TextBox2 selected, in the Properties window set the Value to **=Year(Fields!SaleDate.Value)** using expressions. This groups and displays the data by Year.
4. In the Group Editor, select the **SaleDate** group, and in the Properties window expand the **Group** property node and click the ellipsis button next to the GroupExpressions property to open the **Expressions** dialog.
5. In the **Expressions** dialog that appears, select the group member from the Members list and in the property grid to the right, enter the expression **=Year(Fields!SaleDate.Value)** to group the data by year and then click **OK** to close the dialog.
6. Right-click TextBox2 where **SaleDate** field is added in the Tablix data region, select **Add Column Group**, and then select the **Adjacent Right** option. The adjacent column group is added to the right and is listed under the Group Editor window.
7. Hover over TextBox5 to reveal the field selection adorner, click it to display a list of available fields, and select the **Description** field. This is a column group cell, so selecting a field in it automatically adds a column group.
8. Right-click TextBox5 where **Description** field is added in the Tablix data region, select **Insert Row**, and then select **Outside Group - Above**. The row is added above the new column group and is listed under the Group Editor window.
9. In TextBox7 and TextBox8 above the **SaleDate** and **Description** fields, set the Value property to **Year** and **Media Type**, respectively.
10. Hover over TextBox3 to reveal the field selection adorner, click it to display a list of available fields, and select the **StoreName** field. This is a row group cell, and selecting a field in it automatically adds a row group.
11. Hover over TextBox4 to reveal the field selection adorner, click it to display the list of available fields, and select the **TotalAmount** field.
12. Hover over TextBox6 to reveal the field selection adorner, click it to display the list of available fields, and select the **TotalAmount** field.



Enhancing the appearance of the report

When you preview the report at this point, you will notice the data from the fields is displayed in the Tablix data region. We can enhance the layout of the Tablix data region by setting cell properties in the Properties Window as follows:

1.

Cell	Property Name	Property Value
=Year([SaleDate])	BackgroundColor	WhiteSmoke
	BorderStyle	Solid
	FontWeight	Bold
	TextAlign	Center
Year	BackgroundColor	LightSteelBlue
	BorderStyle	Solid
	FontWeight	Bold
	TextAlign	Center
=[Description]	BackgroundColor	SlateGray
	BorderStyle	Solid
	FontWeight	Bold
	TextAlign	Center
Media Type	BackgroundColor	LightSteelBlue
	BorderStyle	Solid
	FontWeight	Bold
	TextAlign	Center
=[StoreName]	BackgroundColor	WhiteSmoke
	BorderStyle	Solid
	TextAlign	Center
=Sum([TotalAmount])	Format	c
	BorderStyle	Solid
	TextAlign	Center

2. Select the two empty cells in the corner area (top left corner), right-click the selected area, and then select **Merge Cells** option to merge the cells.
3. With the merged cell selected, in the Properties window set the **Value** property to **Sales by Year and Media Type** . This is the heading for the report.
4. In the Tablix data region, select the textbox that contains **Sales by Year and Media Type** text and then go to Properties Window to set the following properties.

Property Name	Property Value
BackgroundColor	LightSteelBlue
BorderStyle	Solid
FontWeight	Bold
TextAlign	Center

5. Select the Tablix data region and go to the Properties window to set the following properties.

Property Name	Property Value
Location	0in, 0in
Size	3in, 1.125in

Viewing the report

- Click the preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

Cell Merging in a Row Group Area in Tablix

This walkthrough illustrates a step-by-step implementation for creating a report which uses the Tablix data region to display store numbers and managers by region and district. The report in this walkthrough demonstrates how Tablix cells with same value in a row group area automatically merge to avoid clutter.

The walkthrough is split into the following activities:

- **Creating an ActiveReports project in Visual Studio**
- **Connecting the report to a data source**
- **Adding a dataset**
- **Creating a layout for the report**
- **Enhancing the appearance of the report**
- **Viewing the report**

> **Note:**
>
> - This walkthrough uses the Reels database. The Reels.mdb file can be downloaded from GitHub: ..\Samples14\Data\Reels.mdb.
> - Although this walkthrough uses RDL reports, you can also implement this using page reports.

When you complete this walkthrough, you will have a layout that looks similar to the following at design time and at run time.

Design-Time Layout

Run-Time Layout

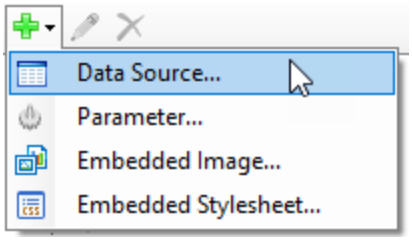
To create an ActiveReports project in Visual Studio

1. Create a new Visual Studio project.
2. In the **New Project** dialog that appears, select **ActiveReports 14 RDL Report Application** and in the Name field, name the file rptTablix.
3. Click **OK** to create a new **ActiveReports 14 RDL Report Application**. By default an RDL report is added to the project.

See Quick Start to a Project for information on adding different report layouts.

To connect a report to a data source

1. In the Report Explorer, right-click the **Data Sources** node and select the **Add Data Source** option or select **Data Source** from the Add button.



2. In the Report Data Source Dialog that appears, select the **General** page and in the Name field, enter a name like ReportData.
3. On this page, create a connection to the Reels database. See Connect to a Data Source for information on connecting to a data source.

To add a dataset

1. In the Report Explorer, right-click the **Data Sources** node and select the **Add Data Set** option or select **Data Set** from the Add button.
2. In the DataSet Dialog that appears, select the **General** page and name the dataset **StoreDetails**. This name appears as a child node of the data source icon in the Report Explorer.
3. On the **Query** page of this dialog, in the Query field enter the following SQL query.

SQL Query
`SELECT Regions.RegionID, Regions.Region, Districts.DistrictID, Districts.District,`

```
Store.StoreName, Person.FirstName, Person.LastName FROM ((Regions INNER JOIN
Districts ON Regions.RegionID = Districts.Region) INNER JOIN Store ON
Districts.DistrictID = Store.DistrictID) INNER JOIN Person ON Store.Manager =
Person.PersonID;
```

4. Click the **Validate DataSet** icon at the top right hand corner above the Query box to validate the query.

5. Click **OK** to close the dialog. Your data set and queried fields appear as nodes in the Report Explorer.

To create a layout for the report

1. From the toolbox, drag a Tablix data region onto the designer surface of the report.
2. Hover over TextBox3 to reveal the field selection adorner, click it to display a list of available fields, and select the **Region** field. This a row group cell, and selecting a field in it automatically adds a row group.
3. Select TextBox1 above the **Region** row group cell, and in the Properties window set the **Value** property to **Region**. This is the heading for the row group.
4. Right-click TextBox3 where the **Region** field was added in the Tablix data region, select **Add Row Group**, and then select the **Child Group** option. The child group is added to the right of the row header and is listed under the Group Editor window.
5. Hover over TextBox5 to reveal the field selection adorner, click it to display a list of available fields, and select the **District** field. This a row group cell, and selecting a field in it automatically adds a row group.
6. Select Textbox6 above the **District** row group cell, and in the Properties window set the **Value** property to **District**. This is the heading for the row group.
7. Right-click TextBox5 where the **District** field was added in the Tablix data region, select **Add Row Group**, and then select the **Child Group** option. The child group is added next to the new row group and is listed under the Group Editor window.
8. Hover over TextBox7 to reveal the field selection adorner, click it to display a list of available fields, and select the **StoreName** field. This a row group cell, and selecting a field in it automatically adds a row group.
9. Select Textbox8 above the **StoreName** row group cell, and in the Properties window set the **Value** property to **Store**. This is the heading for row group.
10. Right click Textbox8 above the **StoreName** row group cell, select **Insert Column**, and then select **Right**. This adds a new static column.
11. Hover over TextBox9 to reveal the field selection adorner, click it to display a list of available fields, and select the FirstName field.
12. With TextBox9 selected, in the Properties window set the **Value** property to **=Fields!FirstName.Value & " " & Fields!LastName.Value** using expressions.
13. Select Textbox10 above the **=[FirstName] & " " & [LastName]** column cell, and in the Properties window set the **Value** property to **Manager**. This is the heading for the static column.

To enhance the appearance of the report

When you preview the report at this point, you will notice the data from the fields is displayed in the Tablix data region. We can enhance the layout of the Tablix data region by setting cell properties in the Properties Window as follows:

Cell	Property Name	Property Value
=[Region]	BackgroundColor	Gainsboro
	BorderStyle	Solid

	FontWeight	Bold
	TextAlign	Center
	VerticalAlign	Middle
Region	BackgroundColor	Gray
	BorderStyle	Solid
	FontWeight	Bold
	TextAlign	Center
=[District]	BackgroundColor	LightSteelBlue
	BorderStyle	Solid
	FontWeight	Bold
	TextAlign	Center
	VerticalAlign	Middle
District	BackgroundColor	Gray
	BorderStyle	Solid
	FontWeight	Bold
	TextAlign	Center
=[StoreName]	BackgroundColor	WhiteSmoke
	BorderStyle	Solid
	TextAlign	Center
Store	BackgroundColor	Gray
	BorderStyle	Solid
	FontWeight	Bold
	TextAlign	Center
=[FirstName] & " " & [LastName]	BorderStyle	Solid
	TextAlign	Center
Manager	BackgroundColor	Gray
	BorderStyle	Solid
	FontWeight	Bold
	TextAlign	Center

To view the report

- Click the preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

Export

This section contains the following walkthroughs that fall under the Export category.

Custom Web Exporting
: This walkthrough demonstrates how to export your report into several popular formats like PDF, HTML, Excel, Image and Word.

Custom Web Exporting

ActiveReports provides components that allow you to export your reports into several popular formats like PDF, HTML, Excel, Image, Word and XML.

The walkthrough is split up into the following activities:

- Adding an ActiveReport to a Visual Studio project
- Adding code to the Web Form to create the PDF, HTML, Excel, Word, XML and Image Export objects and export a report.
- Running the project

Note: Although this walkthrough uses Page reports, you can also implement this using RDL reports.

To add an ActiveReport to the Visual Studio project

1. Create a new ASP.NET Web Application project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 Page Report** and in the Name field, rename the file as **CustomWebExporting**.
4. Click the **Add** button to open a new fixed page report in the designer.
5. In the Solution Explorer, right-click the References node and select **Add Reference**.
6. In the Add Reference dialog that appears, select the following references and click **OK** to add them to your project.
   ```
   GrapeCity.ActiveReports.Export.Pdf
   GrapeCity.ActiveReports.Export.Html
   GrapeCity.ActiveReports.Export.Excel
   GrapeCity.ActiveReports.Export.Word
   GrapeCity.ActiveReports.Export.Image
   GrapeCity.ActiveReports.Export.Xml
   ```

See Quick Start for information on adding different report layouts.

To add code to the Web Form to create the PDF Export object and export a report

1. Double-click on the design view of the aspx page. This creates an event-handling method for the Page_Load event.
2. Add code like the following to the Page_Load event.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the Page Load event.

```
'Provide the page report you want to render.
Dim report As New GrapeCity.ActiveReports.PageReport(New System.IO.FileInfo(Server.MapPath("") +
"\CustomWebExporting.rdlx"))
Dim reportDocument As New GrapeCity.ActiveReports.Document.PageDocument(report)

'Set the rendering extension and render the report.
Dim pdfRenderingExtension As New GrapeCity.ActiveReports.Export.Pdf.Page.PdfRenderingExtension()
Dim outputProvider As New GrapeCity.ActiveReports.Rendering.IO.MemoryStreamProvider()
reportDocument.Render(pdfRenderingExtension, outputProvider)

Response.ContentType = "application/pdf"
Response.AddHeader("content-disposition", "inline;filename=MyExport.pdf")
Dim ms As New System.IO.MemoryStream()
CType(outputProvider.GetPrimaryStream().OpenStream(), System.IO.MemoryStream).WriteTo(ms)
```

```
Response.BinaryWrite(ms.ToArray())
Response.End()
```

To write the code in C#

C# code. Paste INSIDE the Page Load event.

```
// Provide the page report you want to render.
GrapeCity.ActiveReports.PageReport report = new GrapeCity.ActiveReports.PageReport(new
System.IO.FileInfo(Server.MapPath("") + "\\CustomWebExporting.rdlx"));
GrapeCity.ActiveReports.Document.PageDocument reportDocument = new GrapeCity.ActiveReports.Document.PageDocument(report);

// Set the rendering extension and render the report.
GrapeCity.ActiveReports.Export.Pdf.Page.PdfRenderingExtension pdfRenderingExtension = new
GrapeCity.ActiveReports.Export.Pdf.Page.PdfRenderingExtension();
GrapeCity.ActiveReports.Rendering.IO.MemoryStreamProvider outputProvider = new
GrapeCity.ActiveReports.Rendering.IO.MemoryStreamProvider();
reportDocument.Render(pdfRenderingExtension, outputProvider);

Response.ContentType = "application/pdf";
Response.AddHeader("content-disposition", "inline;filename=MyExport.pdf");
System.IO.MemoryStream ms = new System.IO.MemoryStream();
outputProvider.GetPrimaryStream().OpenStream().CopyTo(ms);
Response.BinaryWrite(ms.ToArray());
Response.End();
```

Note: To use the one-touch printing option, add the following to the code above.

Visual Basic.NET code. Paste INSIDE the Page Load event.

```
' Replace the line reportDocument.Render(pdfRenderingExtension, outputProvider) in the code above with the following

Dim setting As New GrapeCity.ActiveReports.Export.Pdf.Page.Settings()
setting.PrintOnOpen = True
reportDocument.Render(pdfRenderingExtension, outputProvider,setting)
```

C# code. Paste INSIDE the Page Load event.

```
// Replace the line reportDocument.Render(pdfRenderingExtension, outputProvider); in the code above with the following

GrapeCity.ActiveReports.Export.Pdf.Page.Settings setting = new GrapeCity.ActiveReports.Export.Pdf.Page.Settings();
setting.PrintOnOpen = true;
reportDocument.Render(pdfRenderingExtension, outputProvider,setting);
```

Warning: You need to manually license your application in order to use PDF export.

To add code to the Web Form to create the HTML Export object and export a report

1. Double-click on the design view of the aspx page. This creates an event-handling method for the Page_Load event.
2. Add code like the following to the Page_Load event.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the Page Load event.

```
' Provide the page report you want to render.
Dim report As New GrapeCity.ActiveReports.PageReport(New System.IO.FileInfo(Server.MapPath("") +
"\CustomWebExporting.rdlx"))
Dim reportDocument As New GrapeCity.ActiveReports.Document.PageDocument(report)

' Set the rendering extension and render the report.
Dim htmlRenderingExtension As New GrapeCity.ActiveReports.Export.Html.Page.HtmlRenderingExtension()
Dim outputProvider As New GrapeCity.ActiveReports.Rendering.IO.MemoryStreamProvider()
Dim setting As New GrapeCity.ActiveReports.Export.Html.Page.Settings()
setting.Mode = GrapeCity.ActiveReports.Export.Html.Page.RenderMode.Galley
setting.MhtOutput = True

reportDocument.Render(htmlRenderingExtension, outputProvider, setting)

Response.ContentType = "message/rfc822"
```

```
Response.AddHeader("content-disposition", "inline;filename=MyExport.mht")
Dim ms As New System.IO.MemoryStream()
CType(outputProvider.GetPrimaryStream().OpenStream(), System.IO.MemoryStream).WriteTo(ms)
Response.BinaryWrite(ms.ToArray())
Response.End()
```

To write the code in C#

C# code. Paste INSIDE the Page Load event.

```
// Provide the page report you want to render.
GrapeCity.ActiveReports.PageReport report = new GrapeCity.ActiveReports.PageReport(new
System.IO.FileInfo(Server.MapPath("") + "\\CustomWebExporting.rdlx"));
GrapeCity.ActiveReports.Document.PageDocument reportDocument = new GrapeCity.ActiveReports.Document.PageDocument(report);

// Set the rendering extension and render the report.
GrapeCity.ActiveReports.Export.Html.Page.HtmlRenderingExtension htmlRenderingExtension = new
GrapeCity.ActiveReports.Export.Html.Page.HtmlRenderingExtension();GrapeCity.ActiveReports.Rendering.IO.MemoryStreamProvider
outputProvider = new
GrapeCity.ActiveReports.Rendering.IO.MemoryStreamProvider();GrapeCity.ActiveReports.Export.Html.Page.Settings setting = new
GrapeCity.ActiveReports.Export.Html.Page.Settings();
setting.Mode = GrapeCity.ActiveReports.Export.Html.Page.RenderMode.Galley;
setting.MhtOutput = true;

reportDocument.Render(htmlRenderingExtension, outputProvider, setting);

Response.ContentType = "message/rfc822";
Response.AddHeader("content-disposition", "inline;filename=MyExport.html");
System.IO.MemoryStream ms = new System.IO.MemoryStream();
outputProvider.GetPrimaryStream().OpenStream().CopyTo(ms);
Response.BinaryWrite(ms.ToArray());
Response.End();
```

To add code to the Web Form to create the Excel Export

1. Double-click on the design view of the aspx page. This creates an event-handling method for the Page_Load event.
2. Add code like the following to the Page_Load event.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the Page Load event.

```
' Provide the page report you want to render.
Dim report As New GrapeCity.ActiveReports.PageReport(New System.IO.FileInfo(Server.MapPath("") +
"\CustomWebExporting.rdlx"))
Dim reportDocument As New GrapeCity.ActiveReports.Document.PageDocument(report)

' Provide settings for your rendering output.
Dim excelSetting As New GrapeCity.ActiveReports.Export.Excel.Page.ExcelRenderingExtensionSettings()
excelSetting.FileFormat = GrapeCity.ActiveReports.Export.Excel.Page.FileFormat.Xlsx
Dim setting As GrapeCity.ActiveReports.Extensibility.Rendering.ISettings = excelSetting

' Set the rendering extension and render the report.
Dim excelRenderingExtension As New GrapeCity.ActiveReports.Export.Excel.Page.ExcelRenderingExtension()
Dim outputProvider As New GrapeCity.ActiveReports.Rendering.IO.MemoryStreamProvider()
reportDocument.Render(excelRenderingExtension, outputProvider, setting.GetSettings())

Response.ContentType = "application/vnd.ms-excel"
Response.AddHeader("content-disposition", "inline;filename=MyExport.xls")
Dim ms As New System.IO.MemoryStream()
outputProvider.GetPrimaryStream().OpenStream().CopyTo(ms)
Response.BinaryWrite(ms.ToArray())
Response.[End]()
```

To write the code in C#

C# code. Paste INSIDE the Page Load event.

```
// Provide the page report you want to render.
```

```
GrapeCity.ActiveReports.PageReport report = new GrapeCity.ActiveReports.PageReport(new
System.IO.FileInfo(Server.MapPath("") + "\\CustomWebExporting.rdlx"));
GrapeCity.ActiveReports.Document.PageDocument reportDocument = new GrapeCity.ActiveReports.Document.PageDocument(report);

// Provide settings for your rendering output.
GrapeCity.ActiveReports.Export.Excel.Page.ExcelRenderingExtensionSettings excelSetting = new
GrapeCity.ActiveReports.Export.Excel.Page.ExcelRenderingExtensionSettings();
excelSetting.FileFormat = GrapeCity.ActiveReports.Export.Excel.Page.FileFormat.Xlsx;
GrapeCity.ActiveReports.Extensibility.Rendering.ISettings setting = excelSetting;

// Set the rendering extension and render the report.
GrapeCity.ActiveReports.Export.Excel.Page.ExcelRenderingExtension excelRenderingExtension = new
GrapeCity.ActiveReports.Export.Excel.Page.ExcelRenderingExtension();
GrapeCity.ActiveReports.Rendering.IO.MemoryStreamProvider outputProvider = new
GrapeCity.ActiveReports.Rendering.IO.MemoryStreamProvider();
reportDocument.Render(excelRenderingExtension, outputProvider, setting.GetSettings());

Response.ContentType = "application/vnd.ms-excel";
Response.AddHeader("content-disposition", "inline;filename=MyExport.xls");
System.IO.MemoryStream ms = new System.IO.MemoryStream();
outputProvider.GetPrimaryStream().OpenStream().CopyTo(ms);
Response.BinaryWrite(ms.ToArray());
Response.End();
```

To add code to the Web Form to create the Word Export object and export a report

1. Double-click on the design view of the aspx page. This creates an event-handling method for the Page_Load event.
2. Add code like the following to the Page_Load event.

> **Note:** To export your report in .Doc format, change the **FileFormat** property option from .Docx to .Doc format as depicted below.
> `wordSetting.FileFormat = GrapeCity.ActiveReports.Export.Word.Page.FileFormat.Doc`

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the Page Load event.

```
' Provide the page report you want to render
Dim report As New GrapeCity.ActiveReports.PageReport(New System.IO.FileInfo(Server.MapPath("") +
"\CustomWebExporting.rdlx"))
Dim reportDocument As New GrapeCity.ActiveReports.Document.PageDocument(report)

' Set the rendering extension and render the report
Dim wordRenderingExtension As New GrapeCity.ActiveReports.Export.Word.Page.WordRenderingExtension()
Dim outputProvider As New GrapeCity.ActiveReports.Rendering.IO.MemoryStreamProvider()

' Set FileFormat property to .Docx
Dim wordSetting As New GrapeCity.ActiveReports.Export.Word.Page.Settings()
wordSetting.FileFormat = GrapeCity.ActiveReports.Export.Word.Page.FileFormat.OOXML

reportDocument.Render(wordRenderingExtension, outputProvider, wordSetting)
Response.ContentType = "application/msword"
Response.AddHeader("content-disposition", "inline;filename=MyExport.docx")
Dim ms As New System.IO.MemoryStream()
CType(outputProvider.GetPrimaryStream().OpenStream(), System.IO.MemoryStream).WriteTo(ms)
Response.BinaryWrite(ms.ToArray())
Response.End()
```

To write the code in C#

C# code. Paste INSIDE the Page Load event.

```
// Provide the page report you want to render
GrapeCity.ActiveReports.PageReport report = new GrapeCity.ActiveReports.PageReport(new
System.IO.FileInfo(Server.MapPath("") + "\\CustomWebExporting.rdlx"));
GrapeCity.ActiveReports.Document.PageDocument reportDocument = new GrapeCity.ActiveReports.Document.PageDocument(report);

// Set the rendering extension and render the report
GrapeCity.ActiveReports.Export.Word.Page.WordRenderingExtension wordRenderingExtension = new
```

```
GrapeCity.ActiveReports.Export.Word.Page.WordRenderingExtension();
GrapeCity.ActiveReports.Rendering.IO.MemoryStreamProvider outputProvider = new
GrapeCity.ActiveReports.Rendering.IO.MemoryStreamProvider();

// Set the FileFormat property to .Docx
GrapeCity.ActiveReports.Export.Word.Page.Settings wordSetting = new GrapeCity.ActiveReports.Export.Word.Page.Settings();
wordSetting.FileFormat = GrapeCity.ActiveReports.Export.Word.Page.FileFormat.OOXML;

reportDocument.Render(wordRenderingExtension, outputProvider, wordSetting);
Response.ContentType = "application/msword";
Response.AddHeader("content-disposition", "inline;filename=MyExport.docx");
System.IO.MemoryStream ms = new System.IO.MemoryStream();
outputProvider.GetPrimaryStream().OpenStream().CopyTo(ms);
Response.BinaryWrite(ms.ToArray());
Response.End();
```

To add code to the Web Form to create the Image Export object and export a report

1. Double-click on the design view of the aspx page. This creates an event-handling method for the Page_Load event.
2. Add code like the following to the Page_Load event.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the Page Load event.

```
' Provide the page report you want to render.
Dim report As New GrapeCity.ActiveReports.PageReport(New System.IO.FileInfo(Server.MapPath("") +
"\CustomWebExporting.rdlx"))
Dim reportDocument As New GrapeCity.ActiveReports.Document.PageDocument(report)

' Set the rendering extension and render the report.
Dim imageRenderingExtension As New GrapeCity.ActiveReports.Export.Image.Page.ImageRenderingExtension()
Dim outputProvider As New GrapeCity.ActiveReports.Rendering.IO.MemoryStreamProvider()
Dim setting As New GrapeCity.ActiveReports.Export.Image.Page.Settings()
setting.ImageType = GrapeCity.ActiveReports.Export.Image.Page.Renderers.ImageType.JPEG
reportDocument.Render(imageRenderingExtension, outputProvider, setting)

Response.ContentType = "image/jpeg"
Response.AddHeader("content-disposition", "inline;filename=MyExport.jpg")
Dim ms As New System.IO.MemoryStream()

' Get the first page of the report
CType(outputProvider.GetSecondaryStreams()(0).OpenStream(), System.IO.MemoryStream).WriteTo(ms)
Response.BinaryWrite(ms.ToArray())
Response.End()
```

To write the code in C#

C# code. Paste INSIDE the Page Load event.

```
// Provide the page report you want to render.
GrapeCity.ActiveReports.PageReport report = new GrapeCity.ActiveReports.PageReport(new
System.IO.FileInfo(Server.MapPath("") + "\\CustomWebExporting.rdlx"));
GrapeCity.ActiveReports.Document.PageDocument reportDocument = new GrapeCity.ActiveReports.Document.PageDocument(report);

// Set the rendering extension and render the report.
GrapeCity.ActiveReports.Export.Image.Page.ImageRenderingExtension imageRenderingExtension = new
GrapeCity.ActiveReports.Export.Image.Page.ImageRenderingExtension();
GrapeCity.ActiveReports.Rendering.IO.MemoryStreamProvider outputProvider = new
GrapeCity.ActiveReports.Rendering.IO.MemoryStreamProvider();
GrapeCity.ActiveReports.Export.Image.Page.Settings setting = new GrapeCity.ActiveReports.Export.Image.Page.Settings();
setting.ImageType = GrapeCity.ActiveReports.Export.Image.Page.Renderers.ImageType.JPEG;

reportDocument.Render(imageRenderingExtension, outputProvider, setting);

Response.ContentType = "image/jpeg";
Response.AddHeader("content-disposition", "inline;filename=MyExport.jpg");
System.IO.MemoryStream ms = new System.IO.MemoryStream();
```

```
// Get the first page of the report
outputProvider.GetSecondaryStreams()[0].OpenStream().CopyTo(ms);
Response.BinaryWrite(ms.ToArray());
Response.End();
```

To add code to the Web Form to create the XML Export object and export a report.

1. Double-click on the design view of the aspx page. This creates an event-handling method for the Page_Load event.
2. Add code like the following to the Page_Load event.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the Page Load event.

```
' Provide the page report you want to render.
Dim report As New GrapeCity.ActiveReports.PageReport(New System.IO.FileInfo(Server.MapPath("") +
"\CustomWebExporting.rdlx"))
Dim reportDocument As New GrapeCity.ActiveReports.Document.PageDocument(report)

' Set the rendering extension and render the report.
Dim xmlRenderingExtension As New GrapeCity.ActiveReports.Export.Xml.Page.XmlRenderingExtension()
Dim outputProvider As New GrapeCity.ActiveReports.Rendering.IO.MemoryStreamProvider()
reportDocument.Render(xmlRenderingExtension, outputProvider)

Response.ContentType = "application/xml"
Response.AddHeader("content-disposition", "inline;filename=MyExport.xml")
Dim ms As New System.IO.MemoryStream()
outputProvider.GetPrimaryStream().OpenStream().CopyTo(ms)
Response.BinaryWrite(ms.ToArray())
Response.[End]()
```

To write the code in C#

C# code. Paste INSIDE the Page Load event.

```
// Provide the page report you want to render.
GrapeCity.ActiveReports.PageReport report = new GrapeCity.ActiveReports.PageReport(new
System.IO.FileInfo(Server.MapPath("") + "\\CustomWebExporting.rdlx"));
GrapeCity.ActiveReports.Document.PageDocument reportDocument = new GrapeCity.ActiveReports.Document.PageDocument(report);

// Set the rendering extension and render the report.
GrapeCity.ActiveReports.Export.Xml.Page.XmlRenderingExtension xmlRenderingExtension = new
GrapeCity.ActiveReports.Export.Xml.Page.XmlRenderingExtension();
GrapeCity.ActiveReports.Rendering.IO.MemoryStreamProvider outputProvider = new
GrapeCity.ActiveReports.Rendering.IO.MemoryStreamProvider();
reportDocument.Render(xmlRenderingExtension, outputProvider);

Response.ContentType = "application/xml";
Response.AddHeader("content-disposition", "inline;filename=MyExport.xml");
System.IO.MemoryStream ms = new System.IO.MemoryStream();
outputProvider.GetPrimaryStream().OpenStream().CopyTo(ms);
Response.BinaryWrite(ms.ToArray());
Response.End();
```

To run the project

Press **F5** to run the project.

Preview

This section contains the following walkthroughs that fall under the Preview category.

Drilldown Reports

This walkthrough demonstrates how to create a drilldown report using the Hidden and ToggleItem properties.

Drill-Through Reports

This walkthrough demonstrates how to create a drill-through link to another report containing details about the linked item.

Parameterized Reports

This walkthrough demonstrates how to create a report with multivalue parameters and an option to select all of the data.

Reports with Bookmarks

This walkthrough demonstrates how to set up bookmarks and links in a report.

Reports with TableOfContents

This walkthrough demonstrates how to create a report that includes the Table of Contents (ToC) control and displays the document map on a report page.

Drilldown Reports

This walkthrough expands upon the report created in the Master Detail Reports walkthrough. If you have not created the Master Detail report (CustomerOrders.rdlx) already, please do so before continuing.

This walkthrough illustrates how to create a drilldown report using the Hidden and ToggleItem properties.

The walkthrough is split up into the following activities:

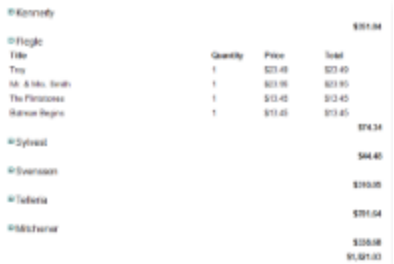
- Opening the Master Detail Report
- Hiding table rows and setting a toggle item
- Viewing the report

> **Note:** This walkthrough uses the **Customer** table from the Reels database. The Reels.mdb file can be downloaded from GitHub: ..\Samples14\Data\Reels.mdb.

When you complete this walkthrough you get a layout that looks similar to the following at run time.

Design-Time Layout

Run-Time Layout



To open the report in Visual Studio

1. Open the Master Detail Report project in Visual Studio.
2. In the Visual Studio Solution Explorer, double-click **CustomerOrders.rdlx**.

To hide table rows and set a toggle item

1. In the designer, click inside the table to display the column and row handles along the top and left sides of the table.
2. Select the second group header row containing the static labels (**Title**, **Quantity**, **Price** and **Total**) by clicking the row handle to the left of it.
3. Hold the CTRL key and select the Detail row containing field expressions to add to the selection.
4. In the Properties window, expand the **Visibility** property and set its properties as follows.

Property Name	Property Value
Visibility Hidden	True
Visibility Toggle Item	TextBox10 (the textbox containing the expression =First(Fields!LastName.Value))

To view the report

- Click the preview tab to view the report at design time.

1. Click the Preview tab of the report designer.
2. Click the **+** icon next to a customer to display order details for that customer.
3. Click the **-** icon to hide the details.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

Drill-Through Reports

The following procedures illustrate how to create a drill-through link to another report containing details about the linked item.

The walkthrough is split into the following activities:

- Creating a main report
- Connecting the main report to a data source and adding a dataset
- Adding controls to the main report to contain data
- Creating a detail report
- Connecting the detail report to a data source
- Adding a dataset with a parameter
- Creating a dataset to populate the parameter values
- Adding a report parameter
- Adding controls to the detail report to contain data
- Adding a drill-through link in the main report
- Viewing the report

Note:

- This walkthrough uses tables from the Reels database. The Reels.mdb file can be downloaded from GitHub: ..\Samples14\Data\Reels.mdb.
- Although this walkthrough uses Page reports, you can also implement this using RDL reports.

When you complete this walkthrough you get a layout that looks similar to the following at run time.

Run-Time Layout (main report)

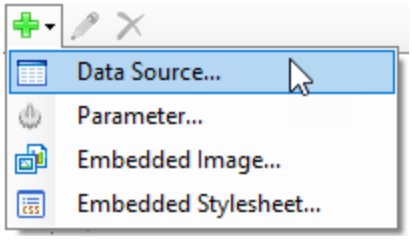
Run-Time Layout (detail report)

To create the main report

1. Create a new Visual Studio project.
2. From the Visual Studio **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 Page Report** and in the Name field, rename the file as **MainReport.rdlx**.
4. Click the **Add** button to open a new page report in the designer.

To connect the main report to a data source and add a dataset

1. In the Report Explorer, right-click the Data Sources node and select the **Add Data Source** option or select **Data Source** from the Add button.



2. In the Report Data Source Dialog that appears, select the **General** page and in the Name field, enter a name like MainReportData.

3. On this page, create a connection to the Reels database. See Connect to a Data Source for information on connecting to a data source.
4. In the Report Explorer, right-click the data source node and select the **Add Data Set** option.
5. In the DataSet Dialog that appears, select the **General** page and enter the name of the dataset as Movie. This name appears as a child node to the data source icon in the Report Explorer.
6. On the **Query** page of this dialog, in the Query field enter the following SQL query.

SQL Query
`SELECT * FROM Movie ORDER BY MovieID ASC`

7. Click the **Validate DataSet** icon at the top right hand corner above the Query box to validate the query.

8. Click **OK** to close the dialog. Your data set and queried fields appear as nodes in the Report Explorer.

To create a layout for the main report

1. In the Visual Studio toolbox, go to the **ActiveReports 14 Page Report** tab and drag a TextBox control onto the design surface.
2. Select the TextBox control and go to the Properties window to set the following properties.

Property Name	Property Value
Location	0.75in, 0.125in
Font	Normal, Arial, 18pt, Bold
Size	5in, 0.5in
TextAlign	Center
Value	MOVIES INFORMATION

3. From the Visual Studio toolbox, drag a Table data region and place it on the design surface.
4. Select the Table and go to the Properties window to set the following properties.

Property Name	Property Value
Location	0in, 1.125in
FixedSize (only for Page reports)	6.5in, 7in
BorderStyle	Solid
RepeatHeaderOnNewPage	True
Size	6.5in, 0.75in

5. In the Table data region, place your mouse over the cells of the table details row to display the field selection adorner.
6. Click the adorner to show a list of available fields from the DataSet and add the following fields to the cells of the table details row.

Cell	Field
Left Cell	MovieID
Middle Cell	Title
Right Cell	YearReleased

This automatically places an expression in the details row and simultaneously places a static label in the header row of the same column.

> **Tip:** You can also directly drag fields from the Report Explorer onto the textbox cells of the Table data region.

7. Select the following table rows and go to the Properties window to set their properties.

Table Header

Property Name	Property Value
BorderStyle	Solid
Font	Normal, Arial, 12pt, Bold
TextAlign	Center

Table Details

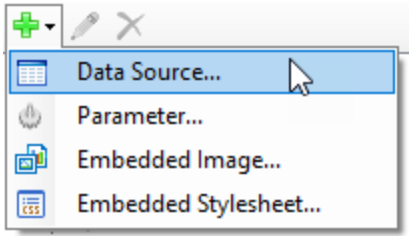
Property Name	Property Value
BorderStyle	Solid
Font	Normal, Arial, 10pt, Bold
TextAlign	Center

To create the detail report

1. From the Visual Studio **Project** menu, select **Add New Item**.
2. In the Add New Item dialog that appears, select **ActiveReports 14 Page Report** and in the Name field, rename the file as **MovieDetails.rdlx**.
3. Click the **Add** button to open a new page report in the designer.

To connect the detail report to a data source

1. In the Report Explorer, right-click the Data Sources node and select the **Add Data Source** option or select **Data Source** from the Add button.



2. In the Report Data Source Dialog that appears, select the **General** page and in the Name field, enter a name like ReportData.
3. On this page, create a connection to the Reels database. See Connect to a Data Source for information on connecting to a data source.

To add a dataset to populate the parameter values

1. In the Report Explorer, right-click the data source node and select the **Add Data Set** option.
2. In the DataSet Dialog that appears, select the **General** page and enter the name of the dataset as **MovieTitles**. This name appears as a child node to the data source icon in the Report Explorer.

3. On the **Query** page of this dialog, in the **Query** field enter the following SQL query.

SQL Query
`SELECT MovieID, Title FROM Movie ORDER BY Title ASC`

4. Click the **Validate DataSet** icon at the top right hand corner above the Query box to validate the query.

5. Click **OK** to close the dialog. Your data set and queried fields appear as nodes in the Report Explorer.

To add a parameter to the report

1. In the Report Explorer, select the Parameters node.
2. Right-click the node and select **Add Parameter** to open the Report - Parameters dialog.
3. Set properties in the following fields below the parameters list.

   In the **General** tab:

   - Name: MovieID
   - DataType: Integer

   In the **Available Values** tab select From query:

   - DataSet: MovieTitles
   - Value: MovieID
   - Label: Title
4. Click **OK** to close the dialog and add the parameter to the collection. This parameter appears under the Parameters node in the Report Explorer.

To add a dataset with a parameter

1. In the Report Explorer, right-click the data source node and select the **Add Data Set** option.
2. In the DataSet Dialog that appears, select the **General** page and enter the name of the dataset as MovieInfo. This name appears as a child node to the data source icon in the Report Explorer.
3. On the **Parameters** page under **Parameter Name** enter MovieID.
4. Under **Value** enter `=Parameters!MovieID.Value`
5. On the **Query** page of this dialog, in the Query field enter the following SQL query.

SQL Query
`Select * from MovieCastInformation`

6. Click the **Validate DataSet** icon at the top right hand corner above the Query box to validate the query.

7. Click **OK** to close the dialog. Your data set and queried fields appear as nodes in the Report Explorer.
8. In an Page report, set the Dataset name in the FixedPage dialog > General tab to MovieInfo. For more information, see Fixed Page Dialog.

> **Caution**: In an Page report, you may get an error if the Dataset name for the FixedPage is not be specified explicitly.

To create a layout for the detail report

1. Click the gray area below the design surface to select the report.
2. Go to the Properties window, expand the **PageSize** property and set the **Width** to 8.5in and **Height** to 3in.
3. From the toolbox, drag a List control onto the design surface and in the Properties window, set the following

properties:

Property Name	Property Value
DataSetName	MovieInfo
Location	0in, 0in
Name	MovieList
Size	6.5in, 1in
FixedSize (only for Page reports)	6.5in, 1in

4. With the List control selected, at the bottom of the Properties Window, select the **Property dialog** command.
5. In the **List** dialog that appears, on the **Detail Grouping** page, set the **Group on: Expression** to `=Fields!MovieID.Value`.
6. Click **OK** to close the dialog.
7. From the Report Explorer, go to the MovieInfo dataset and drag the following five fields onto the **MovieList** data region. In the properties window, set their properties as indicated.

Title

Property Name	Property Value
Name	MovieTitle
Location	0in, 0in
Size	6.5in, 0.375in
TextAlign	Center
FontSize	14pt

YearReleased

Property Name	Property Value
Name	YearReleased
Location	1in, 0.375in
Size	0.75in, 0.25in
TextAlign	Left

MPAA

Property Name	Property Value
Name	MPAA
Location	6in, 0.375in
Size	0.5in, 0.25in

UserRating

Property Name	Property Value
Name	UserRating
Location	1in, 0.625in
Size	0.25in, 0.25in
TextAlign	Left

Length

Property Name	Property Value
Name	Length
Location	4.75in, 0.625in
Size	1.75in, 0.25in
TextAlign	Left
Value	=Fields!Length.Value & " minutes"

Note: When you drag and drop fields from a dataset in the Report Explorer onto the design surface, these fields are automatically converted to Textbox controls that you can modify by setting the control properties in the Properties Window.

8. From the Report Explorer, drag four TextBox controls onto the **MovieList** data region and in the properties window, set their properties as indicated.

TextBox1

Property Name	Property Value
Location	0in, 0.375in
Size	1in, 0.25in
Name	ReleaseLabel
Value	Released in:
FontWeight	Bold

TextBox2

Property Name	Property Value
Location	3.625in, 0.375in
Size	1.875in, 0.25in
Name	MPAALabel
Value	The MPAA rated this film:
FontWeight	Bold

TextBox3

Property Name	Property Value
Location	0in, 0.625in
Size	1in, 0.25in
Name	UserRatingLabel
Value	User rating:
FontWeight	Bold

TextBox4

Property Name	Property Value
Location	4.125in, 0.625in
Size	0.625in, 0.25in
Name	LengthLabel
Value	Length:
FontWeight	Bold

To add a drill-through link to the main report

1. Switch to the Designer with the MainReport.rdlx.
2. On the design surface, select the cell containing the **Title** field inside the table details row and at the bottom of the Properties Window, click the Property dialog command.
3. In the **Textbox** dialog that appears, go to the **Navigation** page.
4. Under **Action**, select **Jump to report** and set the report name MovieDetails.rdlx.
5. Under **Jump to report** set the **Name** of the parameter to MovieID.

   > ⚠ **Caution:** The parameter name must exactly match the parameter in the target report.

6. Set the **Value** to `=Fields!MovieID.Value`.
7. Click **OK** to close the dialog.

To view the report

Open the report in the Viewer. See Windows Forms Viewer for further information.

Parameterized Reports

You can create a parameterized report with ActiveReports and provide the ability to select multiple values for those who want to view data for several items.

This walkthrough illustrates how to create a report with multi-value parameters and an option to select all of the data.

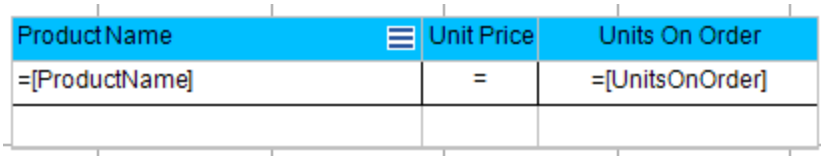
The walkthrough is split up into the following activities:

- Adding an ActiveReport to a Visual Studio project
- Connecting the report to a data source
- Adding a Dataset with a parameter
- Creating a Dataset to populate the parameter values
- Adding a Report Parameter
- Adding controls to the report to contain data
- Viewing the report

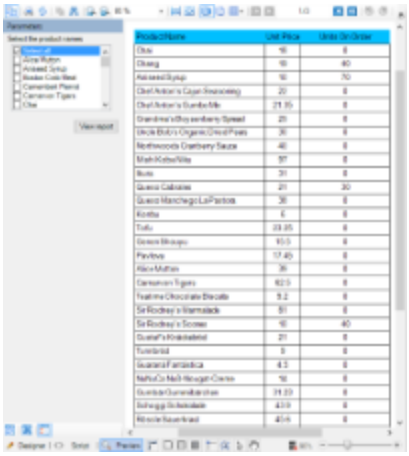
> **Note:**
>
> - This walkthrough uses the **Products** table from the Nwind database. The NWIND.mdb file can be downloaded from GitHub: ..\Samples14\Data\NWIND.mdb.
> - Although this walkthrough uses RDL report, you can also implement this using Page report.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout



Run-Time Layout



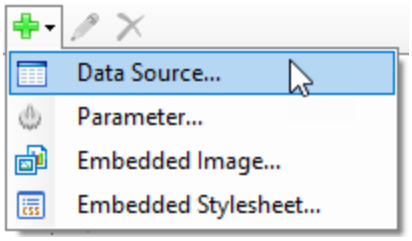
To add an ActiveReport to the Visual Studio project

1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 RDL Report** and in the Name field, rename the file as **Product Details**.
4. Click the **Add** to open a new rdl report in the designer.

See Quick Start for information on adding different report layouts.

To connect the report to a data source

1. In the Report Explorer, right-click the Data Sources node and select the **Add Data Source** option or select **Data Source** from the Add button.



2. In the Report Data Source Dialog that appears, select the **General** page and in the Name field, enter a name like **ReportData**.
3. On this page, create a connection to the Nwind database. See Connect to a Data Source for information on connecting to a data source.

To add a dataset to populate the parameter values

1. In the Report Explorer, right-click the data source node and select the **Add Data Set** option.
2. In the DataSet Dialog that appears, select the **General** page and enter the name of the dataset as **DataSet1**. This name appears as a child node to the data source icon in the Report Explorer.
3. On the **Query** page of this dialog, in the **Query** field enter the following SQL query.

SQL Query
`select distinct productName from Products`

4. Click the **Validate DataSet** icon at the top right hand corner above the Query box to validate the query.
5. Click **OK** to close the dialog. You see the data set, **DataSet1**, and the field, **productName** in the Report Explorer.

To add a parameter to the report

1. In the Report Explorer, select the Parameters node.
2. Right-click the node and select **Add Parameter** to open the Report - Parameters dialog.
3. In the dialog box that appears, click the **Add(+)** button to add a new parameter in the list.
4. Set properties in the following fields below the parameters list.

   In the **General** tab:

   - Name: ReportParameter1
   - DataType: String
   - Text for prompting users for a value: Select the product name.
   - Select the check box next to **Multivalue** to allow users to select more than one product name from the list.
   - Enter Value for 'Select All': 1

   In the **Available Values** tab, select **From query**:

   - DataSet: DataSet1
   - Value: productName
   - Label: productName

   > **Note:** The name of the parameter you enter must exactly match the name of the parameter in the linked report, and it is case sensitive. You can pass a value from the current report to the parameter in the Value column of the list. If a value is not supplied for an expected parameter in the linked report, or if the parameter names do not match, the linked report will not run.

5. Click **OK** to close the dialog and add the parameter to the collection. The ReportParameter1 parameter appears

under the Parameters node in the Report Explorer.

To add a dataset with a parameter

1. In the Report Explorer, right-click the data source node and select the **Add Data Set** option.
2. In the DataSet Dialog that appears, select the **General** page and enter the name of the dataset as **DataSet2**. This name appears as a child node to the data source icon in the Report Explorer.
3. On the **Parameters** page under **Parameter Name** enter **ReportParameter1**.
4. Under **Value** enter `=Parameters!ReportParameter1.Value`.
5. On the **Parameters** page under **Parameter Name** enter **Parameter1**.
6. Under **Value** enter `=Parameters!ReportParameter1.Value`.
7. On the **Query** page of this dialog, in the **Query** field enter the following SQL query.

SQL Query
`select * from products where ProductName in (?) OR '1' in (?)`

8. Click the **Validate DataSet** icon at the top right hand corner above the Query box to validate the query.
9. Click **OK** to close the dialog. Your data set and queried fields appear as nodes in the Report Explorer.

To create a layout for the report

1. From the toolbox, drag a Table control onto the design surface and in the Properties window, set the following properties.

Property Name	Property Value
Location	0.5in, 0.25in
DataSetName	DataSet2
Size	4.8in, 0.75in

2. In the Report Explorer, from the **DataSet2** dataset drag the following fields into the detail row of the table and set their properties as in the following table.

Field	Column	Width
ProductName	TableColumn1	2.37in
UnitPrice	TableColumn2	0.67in
UnitsOnOrder	TableColumn3	1.62in

3. Static labels with the field names are automatically created in the table header row. To improve the appearance of the report, select the table header row, and set the **BackgroundColor** to DeepSkyBlue.

To view the report

- Click the preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

Reports with Bookmarks

You can assign a bookmark ID to any report control and link a text box or image to it to allow users to easily navigate

between items in the finished report. The bookmark link works like a hyperlink, except that clicking a bookmark link jumps to another page or area of the report instead of to a Web page.

This walkthrough explains the steps involved in setting up bookmarks and links.

The walkthrough is split up into the following activities:

- Adding an ActiveReport to a Visual Studio project
- Connecting the report to an XML data source
- Adding a Dataset
- Adding controls to the report to contain data
- Assigning a bookmark ID to a report control
- Adding a bookmark link to a report control
- Viewing the report

> **Note:**
>
> - This walkthrough uses the **Factbook** sample database.
> - Although this walkthrough uses Page reports, you can also implement this using RDL reports.

> **Note:** If you have already created the report outlined in the Reports with XML Data walkthrough, you can open that report and go directly to the **Assigning a Bookmark ID** section of this walkthrough.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout

Run-Time Layout

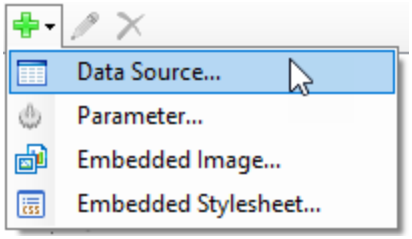
To add an ActiveReport to a Visual Studio project

1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 Page Report** and in the Name field, rename the file as **ExchangeRates.rdlx**.
4. Click the **Add** button to open a new fixed page report in the designer.

See Quick Start for information on adding different report layouts.

To connect the report to a data source

1. In the Report Explorer, right-click the Data Sources node and select the **Add Data Source** option or select **Data Source** from the Add button.



2. In the Report Data Source Dialog that appears, select the **General** page and in the Name field, enter a name like Factbook.
3. On this page, check the **Shared Reference** checkbox.
4. Click the **Browse** button and select Factbook.rdsx. See Connect to a Data Source for information on connecting to a data source.

To add a dataset

1. In the Report Explorer, right-click the data source node and select the **Add Data Set** option or select **Data Set** from the Add button.
2. In the DataSet Dialog that appears, select the **General** page and enter the name of the dataset as ExchangeRates.
3. On the **Query** page, enter the following XML path into the **Query** text box to access data for every country except "World":
```
//country [@name != 'World']
```
4. On the **Fields** page, enter the values in the table below to create fields for your report. Values for XML data fields must be valid XPath expressions.

Field Name	Type	Value
Names	Database Field	@name

Currencies	Database Field	./ExchangeRates/Currency
2004	Database Field	./ExchangeRates/VsUSD2004
2003	Database Field	./ExchangeRates/VsUSD2003
2002	Database Field	./ExchangeRates/VsUSD2002
2001	Database Field	./ExchangeRates/VsUSD2001
2000	Database Field	./ExchangeRates/VsUSD2000

5. Click **OK** to close the dialog. Your data set and queried fields appear as nodes in the Report Explorer.

To add controls to the report

1. From the toolbox, drag a List data region onto the design surface of the report and go to the Properties window to set the **DataSetName** property to ExchangeRates, **Location** property to 0in, 0in and **Size** property to 6.5in, 3in.
2. From the Report Explorer, drag the **Names** field onto the list, center it at the top and go to the Properties window to set the **FontSize** property to 14pt and the **BorderStyle** property to Solid.
3. From the Report Explorer, drag the following fields onto the list with properties set as described in the table below.

Field Name	Property Name
Currencies	Location: 1.125in, 0.5in Size: 2.25in, 0.25in
2004	Location: 4in, 0.875in Size: 1in, 0.25in
2003	Location: 4in, 1.25in Size: 1in, 0.25in
2002	Location: 4in, 1.625in Size: 1in, 0.25in
2001	Location: 4in, 2in Size: 1in, 0.25in
2000	Location: 4in, 2.375in Size: 1in, 0.25in

> **Note**: You will notice that the expressions created for these fields are different than usual. Because Visual Basic syntax does not allow an identifier that begins with a number, any numeric field names must be treated as strings in expressions.

4. From the toolbox, drag a TextBox onto the list and go to the Properties window to set the properties as described in the table below to combine static text with a field value.

Property Name	Property Value
Location	0.125in, 0.875in
Size	3.125in, 0.25in
Value	="Value of " & Fields!Currency.Value & " versus US$ for year:"

Font	Normal, Arial, 10pt, Bold
TextAlign	Right

5. From the toolbox, drag TextBox controls onto the list and go to the Properties window to set the properties as described in the table below to create static labels.

TextBox1

Property Name	Property Value
Location	0.125in, 0.5in
Size	0.75in, 0.25in
FontWeight	Bold
Value	Currency:

TextBox2

Property Name	Property Value
Location	3.375in, 0.875in
Size	0.5in, 0.25in
TextAlign	Right
Value	2004:

TextBox3

Property Name	Property Value
Location	3.375in, 1.25in
Size	0.5in, 0.25in
TextAlign	Right
Value	2003:

TextBox4

Property Name	Property Value
Location	3.375in, 1.625in
Size	1in, 0.25in
TextAlign	Right
Value	2002:

TextBox5

Property Name	Property Value

Location	3.375in, 2in
Size	0.5in, 0.25in
TextAlign	Right
Value	2001:

TextBox6

Property Name	Property Value
Location	3.375in, 2.375in
Size	0.5in, 0.25in
TextAlign	Right
Value	2000:

To assign a bookmark ID to the report control

A bookmark ID marks any report control as something to which a bookmark link can navigate.

1. On the designer surface, select the **Names** textbox at the top of the list and at the bottom of the Properties Window, select the **Property dialog** command.
2. In the **Textbox** dialog that appears, go to the **Navigation** page.
3. On the **Navigation** page, enter **Names** in the Bookmark ID field.

   > **Note:** Every bookmark ID in a report must be a unique string. If you have duplicates, a link to it will navigate only to the first one it finds.

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

To add a bookmark link to the report control

A bookmark link is like a hyperlink that navigates to a report control marked with a bookmark ID instead of to a URL. We will add a text box below the list we already created to display at the bottom of the report. This text box will have a bookmark link to the bookmark ID we created in the last procedure.

1. From the toolbox, drag a text box onto the report below the list and in the Properties window, set the properties as follows.

Property Name	Property Value
Value	Back to Top
Location	0in, 3in
Size	6.5in, .5in
TextAlign	Center

2. At the bottom of the Properties Window, select the **Property dialog** command.
3. In the **Textbox** dialog that appears, go to the **Navigation** page.
4. On the **Navigation** page, select the **Jump to Bookmark** radio button and enter the bookmark ID (**Names**) created in the procedure above.
5. Click **OK** to close the dialog.

To view the report

Open the report in the Viewer. See Windows Forms Viewer for further information.

Reports with TableOfContents

This walkthrough illustrates how to create a report that includes the Table of Contents (ToC) control and displays the document map on a report page.

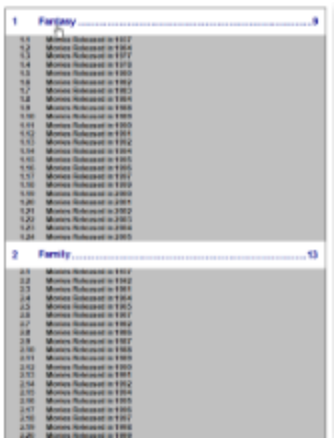
The TableOfContents control allows you to quickly navigate to desired data inside a report. You can use the TableOfContents control to embed the list of contents in the report body for printing and rendering, unlike the Document Map that is only available in the Viewers and cannot be rendered or printed.

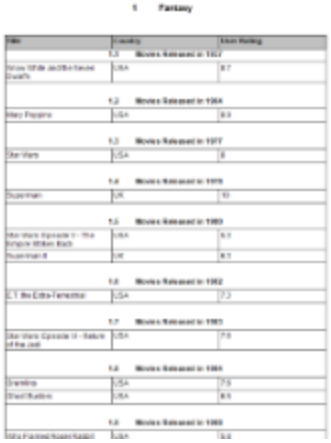
- **To add an ActiveReports to the Visual Studio project**
- **To connect the report to a data source**
- **To add a dataset**
- **To create a layout for the report**
- **To configure the appearance of Table of Contents**
- **To view the report**

When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout

Run-Time Layout





To add an ActiveReports to the Visual Studio project

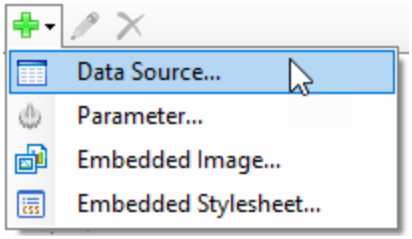
1. Create a new Visual Studio Windows Forms Application project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 RDL Report** and in the Name field, rename the file as **ReportsWithToC.rdlx**.
4. Click the **Add** button to open a new RDL report.

See Quick Start for information on adding different report layouts.

To connect the report to a data source

This walkthrough uses the **Movies** table from the Reels database. The Reels.mdb file can be downloaded from GitHub: ..\Samples14\Data\Reels.mdb.

1. In the Report Explorer, right-click the Data Sources node and select the **Add Data Source** option or select **Data Source** from the Add button.



2. In the Report Data Source Dialog that appears, select the **General** page and in the Name field, enter a name like ReportData.
3. On this page, create a connection to the Reels database. See Connect to a Data Source for information on connecting to a data source.

To add a dataset

1. In the Report Explorer, right-click the data source node and select the **Add Data Set** option or select **Data Set** from the Add button.
2. In the DataSet Dialog that appears, select the **General** page and enter the name of the dataset as **MovieCatalog**. This name appears as a child node to the data source icon in the Report Explorer.
3. On the **Query** page of this dialog, in the **Query** field enter the following SQL query.

SQL Query

```
SELECT Genre.GenreName, Movie.Title, Movie.YearReleased, Movie.UserRating,
Movie.Country FROM Genre INNER JOIN (Movie INNER JOIN MovieGenres ON Movie.MovieID
= MovieGenres.MovieID) ON Genre.GenreID = MovieGenres.GenreID ORDER BY YearReleased
ASC
```

4. Click the **Validate DataSet** icon at the top right hand corner above the Query box to validate the query.

5. Click **OK** to close the dialog. Your data set and queried fields appear as nodes in the Report Explorer.

To create a layout for the report

1. From the toolbox, drag the List control onto the design surface and in the Properties window, set the following properties.

Property Name	Property Value
DataSetName	MovieCatalog
Location	0.25in, 1.875in
Size	6in, 4in
PageBreakAtStart	True **Note:** Applicable for RDL reports only.

2. With the List data region selected, under the Properties window, click the **Property dialog** link to open the List dialog
3. Go to the **Detail Grouping** page and on the General tab, under **Group on**, set the **Expression** field to `=Fields!GenreName.Value`.
4. Click **OK** to close the dialog
5. In the Report Explorer, from the MovieCatalog dataset, drag and drop the **GenreName** field inside the List data region and in the Properties window, set the following properties.

Property Name	Property Value
Location	0.25in, 0.375in
Font	Normal, Arial, 12pt, Bold
TextAlign	Center
Size	5.625in, 0.25in
Label	=Fields!GenreName.Value **Note**: Setting the control's **Label** property adds an entry of the control in the document map.

6. From the toolbox, drag and drop the Table data region inside the List data region and in the Properties window, set the following properties.

Property Name	Property Value
Location	0.125in, 1in
Size	6in, 0.75in
DataSetName	MovieCatalog
BorderStyle	Solid
RepeatHeaderOnNewPage	True

7. In the Table data region, place your mouse over the cells of the table details row to display the field selection

adorner.

8. Click the adorner to show a list of available fields from the MovieCatalog dataset and add the following fields to the cells of the table details row.

Cell	Field
Left Cell	Title
Middle Cell	Country
Right Cell	UserRating

This automatically places an expression in the details row and simultaneously places a static label in the header row of the same column.

> **Tip:** You can also directly drag fields from the Report Explorer onto the textbox cells of the Table data region.

9. Select the table header row using the row handle to the left and in the Properties Window set the following properties.

Property Name	Property Value
Font > FontWeight	Bold
TextAlign	Left
BackgroundColor	Silver
BoderStyle	Solid

10. Select the table detail row using the row handle to the left, and in the Properties Window set the following properties.

Property Name	Property Value
TextAlign	Left
BorderStyle	Solid

11. Right-click the table detail row using the row handle to the left, and select **Insert Group...**
12. In the **Table - Groups** dialog that appears, on the General tab, under **Group on**, set the **Expression** field to `=Fields!YearReleased.Value`.
13. Click **OK** to close the dialog.
14. Select the textboxes inside the table group row using the CTRL key and left mouse button, right-click the selection and select **Merge Cells**.
15. Select the merged cell and in the Properties window, set the following properties.

Property Name	Property Value
Font	Normal, Arial, 10pt, Bold
HeadingLevel	Heading 2 **Note**: Setting the control's **HeadingLevel** property adds an entry of the control in the document map.

TextAlign	Center
Value	="Movies Released in " & Fields !YearReleased.Value

16. From the Visual Studio toolbox, drag a TableOfContents control onto the design surface and in the Properties window, set the following properties.

Property Name	Property Value
Location	0.25in, 0.5in
Size	6in, 0.875in
BorderStyle	Solid

To configure the appearance of Table of Contents

1. With the TableOfContents control selected, select the **Levels (Collection)** property and then click the ellipsis button that appears.
2. In the **LevelDesigner Collection Editor** dialog that appears, under **Members**, use the **Add** button to add Level2 to the TableOfContents.
3. Under **Members**, select **Level1** and click the **Property Pages** button above the LevelDesigner Collection Editor properties grid.
4. In the **Level Properties** dialog that appears, set the following properties.

   Properties

Property Name	Property Value
Font > Size	14
Font > Weight	Bold
Color	DarkBlue
Padding > Left	10pt
Padding > Top	10pt
Padding > Right	10pt
Padding > Bottom	10pt
Fill character	.

5. In the **LevelDesigner Collection Editor**, select the **Level2** entry under **Members** and click the **Property Pages** button above the LevelDesigner Collection Editor Properties grid.
6. In the **Level Properties** dialog that appears, set the following properties.

   Properties

Property Name	Property Value
Font > Weight	Bold
Padding > Left	20pt

Padding > Top	10pt
Padding > Right	0pt
Padding > Bottom	0pt
DisplayPageNumber	False

7. Click **OK** to close the **LevelDesigner Collection Editor** dialog.
8. In the Report Explorer, select the **Report** node and in the Properties window set the following properties:

Properties

Property Name	Property Value
DocumentMap > Source	Labels and Headings
DocumentMap > NumberingStyle	1, 2, 3, 4, 5

To view the report

- Click the preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

Advanced

This section contains the following walkthroughs that fall under the Advanced category.

Reports with Custom Code
: This walkthrough demonstrates how to create a simple report with custom code.

Custom Resource Locator
: This walkthrough is based on the Custom Resource Locator sample and illustrates how to load pictures from the user's **My Pictures** directory.

Custom Data Provider
: This walkthrough demonstrates how to create a solution with projects that create a custom data provider and demonstrate how it pulls data from a comma separated values (CSV) file.

Reports with Custom Code

This walkthrough illustrates how to create a simple report with custom code.

The walkthrough is split up into the following activities:

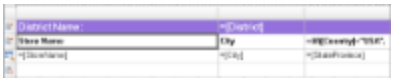
- Adding an ActiveReport to a Visual Studio project
- Connecting the report to a data source
- Adding a dataset
- Adding controls to the report to contain data
- Embedding code in a report and referencing it in a field expression

- Viewing the report

> **Note:**
>
> - This walkthrough uses the **Store** table from the Reels database. The Reels.mdb file can be downloaded from GitHub: ..\Samples14\Data\Reels.mdb.
> - Although this walkthrough uses Page reports, you can also implement this using RDL reports.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout



Run-Time Layout

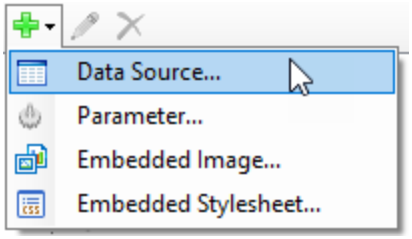
To add an ActiveReport to the Visual Studio project

1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 Page Report** and in the Name field, rename the file as CustomCode.
4. Click the **Add** button to open a new fixed page report in the designer.

See Quick Start for information on adding different report layouts.

To connect the report to a data source

1. In the Report Explorer, right-click the Data Sources node and select the **Add Data Source** option or select **Data Source** from the Add button.



2. In the Report Data Source Dialog that appears, select the **General** page and in the **Name** field, enter a name like ReportData.
3. On this page, create a connection to the Reels database. See Connect to a Data Source for information on connecting to a data source.

To add a dataset

1. In the Report Explorer, right-click the data source node and select the **Add Data Set** option or select **Data Set** from the Add button.
2. In the DataSet Dialog that appears, select the **General** page and enter the name of the dataset as **Districts**. This name appears as a child node to the data source icon in the Report Explorer.
3. On the **Query** page of this dialog, in the **Query** field enter the following SQL query.

SQL Query
`SELECT Store.StoreName, Address.City, Address.Region AS StateProvince, Address.Country, Districts.District FROM Address INNER JOIN (Districts INNER JOIN Store ON Districts.DistrictID = Store.DistrictID) ON Address.AddressID = Store.Address WHERE NOT Districts.DistrictID = 0 ORDER BY Districts.District`

4. Click the **Validate DataSet** icon at the top right hand corner above the Query box to validate the query.

5. Click **OK** to close the dialog. Your data set and queried fields appear as nodes in the Report Explorer.

To create a layout for the report

1. From the toolbox, drag the TextBox control onto the design surface and go to the Properties window to set the following properties:

Property Name	Property Value
Location	0in, 0in
Size	6.5in, 0.5in
TextAlign	Center
FontSize	14pt
Value	District Locations

2. From the toolbox, drag a Table data region onto the design surface and go to the Properties window to set the **DataSetName** property to Districts.
3. Set the following properties for the table:

Property Name	Property Value
Location	0in, 0.5in
FixedSize	6in, 7in **Note**: FixedSize property is set only in Page report.

4. Click inside the table to display the column handles at the top and in the Properties window, set the **Width** property of following columns:

Column	Width
TableColumn1	3in
TableColumn2	1.5in
TableColumn3	1.5in

5. Click inside the table to display the row handles along the left of the table and right-click any of the row handles to select **Insert Group**.
6. In the **Table - Groups** dialog that appears, under **Group on**, select the following expression:
   `=Fields!District.Value`
7. On the same dialog set the **Name** to District and click **OK** to close the dialog. The header and footer rows for the new group appear.
8. In the Report Explorer from the Districts dataset, drag the **District** field into the second column of the group header row of the table. This automatically places an expression in the group header row and simultaneously places a static label in the table header row.
9. In the first column of the group header row, just to the left of the District field, in the Properties window set the **Value** property to **District Name:**.
10. Click the row handle to the left of the group header row to select the entire row and in the Properties window, set the properties as follows:

Property Name	Property Value
FontSize	12pt
FontWeight	Bold
BackgroundColor	MediumPurple
Color	White

11. Right-click any row handle to the left of the table and select **Table Header** to remove the table header.
12. Right-click any row handle to the left of the table and select **Table Footer** to remove the table footer.
13. In the Report Explorer, drag the following fields from the Districts dataset onto the detail row of the table as follows.

Field	Column
StoreName	TableColumn1
City	TableColumn2
StateProvince	TableColumn3

14. Remove the static label **State Province** from the group header for the third column.
15. Right-click the row handle for the group header row and select **Insert Row Below** to add a row for static labels that will appear once for each group.
16. In the new row, enter the following values for static label in table columns.

    TableColumn1

Property Name	Property Value
Value	Store

	Name
FontWeight	Bold

TableColumn2

Property Name	Property Value
Value	City
FontWeight	Bold

TableColumn3

Property Name	Property Value
Value	=iif(Fields!Country.Value="USA", "State", "Province")
FontWeight	Bold

Note: The expression in the third column displays the label "State" when the country is USA, and displays "Province" when it is not.

To embed code in a report and reference it in a field expression

This custom code creates a URL to Yahoo!® Maps for each city in the report.

1. On the **Script** tab of the report, enter the following code to create a URL.

Visual Basic.NET code. Add to the Script tab.

```
Public Function MapLink(ByVal Country, ByVal City, ByVal StateProvince) As String
        Dim Link As String
        Dim _Country As String = Country.ToString()
        Dim _City As String = City.ToString()
        Dim _StateProvince As String = StateProvince.ToString()

        Select Case _Country
                Case "USA"
                  Link = "http://maps.yahoo.com/maps_result?addr=&csz=" & _City &
"%2C+" & _StateProvince & "&country=us&new=1&name=&qty="
                Case "Canada"
                  Link = "http:file://ca.maps.yahoo.com/maps_result?csz=%2C+" &
_StateProvince & "&country=ca"
                Case Else
                  Link = ""
        End Select

        Return Link
End Function
```

> **Note:** Custom code is helpful if you intend to reuse code throughout the report or if code is too complex to use in an expression. Code must be instance based and written in Visual Basic.NET. You can include multiple methods, but if you want to use classes or other .NET languages, create a custom assembly. See Using Script in a Page Report for further details.

To reference embedded code in a field expression

1. On the Designer tab of the report, click the detail cell in the second table column (containing the expression `=Fields!City.Value`) to select it and under the Properties window, click the Property dialog link. This is a command to open the respective control's dialog. See Properties Window for more on how to access commands. .
2. In the Textbox - General dialog that appears, go to the **Navigation** page.
3. Select the radio button next to **Jump to URL**, and enter the following expression in the combo box below it. = `Code.MapLink(Fields!Country.Value, Fields!City.Value, Fields!StateProvince.Value)`
4. Click **OK** to close the dialog and use the code to create a hyperlink for the field.

To view the report

- Click the preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

Custom Resource Locator

Page reports can get resources from your file system using file paths, but sometimes resources are preserved in very specific sources, such as a database. With page reports, you can create a custom resource locator to read any resources that might be required by your reports from any location. This walkthrough is based on the Custom Resource Locator sample and illustrates how to load pictures from the user's **My Pictures** directory.

This walkthrough is split into the following procedures:

- Adding an ActiveReports to the Visual Studio project
- Creating a layout for the report
- Adding the new MyPicturesLocator class
- Creating the PreviewForm that contains the Viewer control to view the report.

> **Note:** Although this walkthrough uses Page reports, you can also implement this using RDL reports.

When you complete this walkthrough you get a layout that looks similar to the following at run time.

To add an ActiveReports to the Visual Studio project

1. Create a new Visual Studio Windows Forms Application project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 Page Report** and in the Name field, rename the file as DemoReport.rdlx.
4. Click the **Add** button to open a new fixed page report.

To create a layout for the report

1. From the toolbox, drag an Image control onto the design surface and in the Properties window, set the following properties.

Property Name	Property Value
Name	Image1
Location	0.1in, 0.1in
Size	2.8in, 2.8in
Value	MyPictures:Penguins.jpg

2. From the toolbox, drag another Image control onto the design surface and in the Properties window, set the following properties.

Property Name	Property Value
Name	Image2
Location	3.1in, 0.1in
Size	2.8in, 2.8in
Value	MyPictures:Desert.jpg

3. In the Solution Explorer, select DemoReport.rdlx and in the Properties window, set **Build Action** to **Embedded Resource**.

To add the new MyPicturesLocator class

1. In the Solution Explorer window, right-click on your project name and select **Add** and then **New Item**.
2. In the **Add New Item** dialog that appears, select **Class**.
3. Change the name of the class to **MyPicturesLocator** and click the **Add** button.
4. Replace the existing code with the following code to the new class.

To write the code in Visual Basic.NET

VB code. Paste on TOP

```
Imports System
Imports System.Drawing
Imports GrapeCity.ActiveReports.Extensibility
Imports System.Globalization
Imports System.IO
Imports System.Runtime.InteropServices
```

VB code. Paste INSIDE the class

```
 Inherits ResourceLocator

    Private Const UriSchemeMyImages As String = "MyPictures:"

    ' Obtain and return the resource.
    Public Overrides Function GetResource(resourceInfo As ResourceInfo) As Resource
```

```
        Dim name As String = resourceInfo.Name
        If name Is Nothing OrElse name.Length = 0 Then
            Throw New ArgumentException("The name of resource to be obtained should be non-
empty string.", "name")
        End If
        Dim uri As New Uri(name)
        Dim stream As Stream = GetPictureFromSpecialFolder(name)
        If stream Is Nothing Then
            stream = New MemoryStream()
        End If
        Return New Resource(stream, uri)
    End Function

    ' Returns the specified image from Public Pictures folder.
    Private Shared Function GetPictureFromSpecialFolder(path As String) As Stream
        Dim startPathPos As Integer = UriSchemeMyImages.Length
        If startPathPos >= path.Length Then
            Return Nothing
        End If
        Dim pictureName As String = path.Substring(startPathPos)
        Dim myPicturesPath As String = Environment.GetEnvironmentVariable("public") &
"\Pictures"
        If Not myPicturesPath.EndsWith("\") Then
            myPicturesPath += "\"
        End If
        Dim picturePath As String = System.IO.Path.Combine(myPicturesPath, pictureName)
        If Not File.Exists(picturePath) Then
            Return Nothing
        End If
        Dim stream As New MemoryStream()
        Try
            Dim picture As Image = Image.FromFile(picturePath)
            picture.Save(stream, picture.RawFormat)
            stream.Position = 0
        Catch generatedExceptionName As OutOfMemoryException
            ' The file is not valid image, or GDI+ doesn't support such images.
            Return Nothing
        Catch generatedExceptionName As ExternalException
            Return Nothing
    End Try
    Return stream
End Function
```

To write the code in C#

C# code. Paste on TOP

```
using System;
using System.Drawing;
using System.Globalization;
using System.IO;
using System.Runtime.InteropServices;
using System.Windows.Forms;
using GrapeCity.ActiveReports.Extensibility;
```

```
using your_project_name.Properties;
```

C# code. Paste BELOW the Using statements

```
namespace your_project_name
{
    // Look for the resources in My Pictures folder.
    internal sealed class MyPicturesLocator : ResourceLocator
    {
        private const string UriSchemeMyImages = "MyPictures:";
        // Obtain and return the resource.
        public override Resource GetResource(ResourceInfo resourceInfo)
        {
            string name = resourceInfo.Name;
            if (name == null || name.Length == 0)
            {
                throw new ArgumentException("The name of resource to be obtained should be
non-empty string.", "name");
            }
            Uri uri = new Uri(name);
            Stream stream = GetPictureFromSpecialFolder(name);
            if (stream == null)
            {
               stream = new MemoryStream();
            }
            return new Resource(stream, uri);
        }
        // Returns the specified image from Public Pictures folder.
        private static Stream GetPictureFromSpecialFolder(string path)
        {
            int startPathPos = UriSchemeMyImages.Length;
            if (startPathPos >= path.ToString().Length)
            {
                return null;
            }
            string pictureName = path.ToString().Substring(startPathPos);
            string myPicturesPath = Environment.GetEnvironmentVariable("public") + \\Pictures;
            if (!myPicturesPath.EndsWith("\\")) myPicturesPath += "\\";
            string picturePath = Path.Combine(myPicturesPath, pictureName);
                        if (!File.Exists(picturePath)) return null;
            MemoryStream stream = new MemoryStream();
            try
            {
                Image picture = Image.FromFile(picturePath);
                picture.Save(stream, picture.RawFormat);
                stream.Position = 0;
            }
            catch (OutOfMemoryException) // The file is not valid image, or GDI+ doesn't
support such images.
            {
                return null;
            }
            catch (ExternalException)
            {
                return null;
```

```
            }
            return stream;
        }
    }
}
```

To create the PreviewForm

1. In the Solution Explorer, select the Form1 in the Design view and in the Properties window, set the properties as follows.

Property Name	Property Value
Name	PreviewForm
Text	Preview Form
Size	1015, 770

2. From the Visual Studio toolbox, drag the Viewer control onto the PreviewForm and in the Properties window, set the following properties.

Property Name	Property Value
Name	reportPreview1
Dock	Fill

3. Double-click the PreviewForm to create an instance for the Load event and add the following code.

To write the code in Visual Basic.NET

VB code. Paste BELOW the Import statements

```
Imports GrapeCity.ActiveReports.Document
Imports System.IO
Imports GrapeCity.ActiveReports
```

VB code. Paste INSIDE the Load event

```
Dim reportData As Stream = [GetType]
().Assembly.GetManifestResourceStream("your_project_name.DemoReport.rdlx")
reportData.Position = 0
Dim reader As New StreamReader(reportData)
Dim def As New PageReport(reader)
def.ResourceLocator = New MyPicturesLocator()
Dim runtime As New PageDocument(def)
reportPreview1.ReportViewer.LoadDocument(runtime)
```

To write the code in C#

C# code. Paste BELOW the Using statements

```
using GrapeCity.ActiveReports.Document;
using System.IO;
using GrapeCity.ActiveReports;
```

C# code. Paste INSIDE the Load event

```
string myPicturesPath = Environment.GetFolderPath(Environment.SpecialFolder.MyPictures);
Stream reportData =
GetType().Assembly.GetManifestResourceStream("your_project_name.DemoReport.rdlx");
```

```
reportData.Position = 0;
StreamReader reader = new StreamReader(reportData);
PageReport def = new PageReport(reader);
def.ResourceLocator = new MyPicturesLocator();
PageDocument runtime = new PageDocument(def);
reportPreview1.ReportViewer.LoadDocument(runtime);
```

4. Press **F5** to run the project.

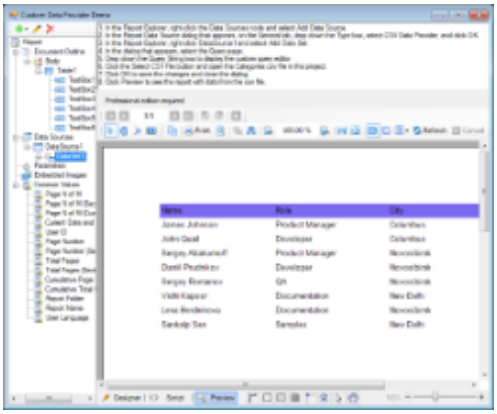
Custom Data Provider

A custom data provider allows you to use non-traditional data sources in your page reports, both at run time and at design time. This walkthrough illustrates how to create a solution with projects that create a custom data provider and demonstrate how it pulls data from a comma separated values (CSV) file.

This walkthrough is split into the following activities:

- Creating a Designer project to demonstrate the custom data provider
- Configuring the project to use a custom data provider
- Adding a report to show the data
- Adding a second project to contain the custom data provider
- Adding a button to the query editor

When you complete this walkthrough, you will have a designer pre-loaded with a report that pulls data from a CSV file and looks like the following.



To create a Designer project to demonstrate the custom data provider

1. In Visual Studio, create a Windows Forms project and name it **CustomDataProviderDemo**.
2. From the Visual Studio toolbox ActiveReports 14 tab, drag a ReportExplorer and drop it onto the default Windows form, resizing the form to a comfortable working area.
3. In the Properties window, set the **Dock** property of the ReportExplorer control to **Left**.
4. From the toolbox Common Controls tab, drag a RichTextBox control onto the form and set the **Dock** property to **Top**.
5. Add the following text to the **Text** property. (Drop down the box to ensure that all of the lines of text are added.)

Text. Paste in the Text property of the RichTextBox.

```
1. In the Report Explorer, right-click the Data Sources node and select Add Data
Source.
2. In the Report Data Source dialog that appears, on the General tab, drop down the
```

```
Type box, select CSV Data Provider, and click OK.
3. In the Report Explorer, right-click DataSource1 and select Add Data Set.
4. In the dialog that appears, select the Query page.
5. Drop down the Query String box to display the custom query editor.
6. Click the Select CSV File button and open the Categories.csv file in this
project.
7. Click OK to save the changes and close the dialog.
8. Click Preview to see the report with data from the csv file.
```

6. From the toolbox ActiveReports 14 tab, drag a Designer control and drop it on the empty part of the form.
7. Set the **Dock** property to **Fill**, then right-click the Designer control on the form and select **Bring to front**.
8. Select the ReportExplorer control and in the Properties window, drop down the **ReportDesigner** property and select **Designer1**.
9. Double-click on the form's title bar to create a form Load event, and add code like the following above the class.

Visual Basic code. Paste above the class.

```
Imports System.Xml
Imports System.IO
Imports GrapeCity.ActiveReports.Design
```

C# code. Paste above the class.

```
using System.Xml;
using System.IO;
using GrapeCity.ActiveReports.Design;
```

10. Add the following code to the form Load event.

Visual Basic code. Paste inside the form Load event.

```
Using reportStream = File.OpenRead("DemoReport.rdlx")
        Using reader = XmlReader.Create(reportStream)
                Designer1.LoadReport(reader, DesignerReportType.Page)
        End Using
End Using
```

C# code. Paste inside the form Load event.

```
using (var reportStream = File.OpenRead("DemoReport.rdlx"))
 {
     using (var reader = XmlReader.Create(reportStream))
      {
         designer1.LoadReport(reader, DesignerReportType.Page);
      }
 }
```

To configure the project to use a custom data provider

1. In the Solution Explorer, right-click the project and select **Add**, then **New Item**.
2. In the dialog that appears, select **Text File**, name it **GrapeCity.ActiveReports.config**, and click **Add**.
3. In the Solution Explorer, select the new file and in the Properties window, set its **Copy to Output Directory** property to **Copy always**.
4. Paste the following text into the file and save it. (You can safely ignore the warning that the 'Configuration' element is not declared.)

Paste into the config file.

```
<?xml version="1.0" encoding="utf-8" ?>
<Configuration>
  <Extensions>
    <Data>
      <Extension Name="CSV" DisplayName="CSV Data Provider"
        Type="CustomDataProvider.CsvDataProvider.CsvDataProviderFactory,
          CustomDataProvider"
          CommandTextEditorType="CustomDataProvider.CSVDataProvider.QueryEditor,
        CustomDataProvider"/>
    </Data>
     </Extensions>
</Configuration>
```

5. In the Solution Explorer, right-click the project and select **Add**, then **New Item**.
6. In the dialog that appears, select **Text File**, name it **Categories.csv**, and click **Add**.
7. In the Solution Explorer, click to select the file, and in the Properties window, change the **Copy to Output Directory** property to **Copy always**.
8. Paste the following text into the file and save it.

Paste into the text file.

```
EmployeeID(int32),LastName,FirstName,Role,City
1,James,Yolanda,Owner,Columbus
7,Reed,Marvin,Manager,Newton
9,Figg,Murray,Cashier,Columbus
12,Snead,Lance,Store Keeper,Columbus
15,Halm,Jeffry,Store Keeper,Columbus
17,Hames,Alma,Store Keeper,Oak Bay
18,Nicki,Aubrey,Store Keeper,Columbus
24,Cliett,Vikki,Store Keeper,Newton
```

To add a report to show the data from the custom data provider

1. In the Solution Explorer, right-click the project and select **Add**, then **New Item**.
2. In the dialog that appears, select **ActiveReports 14 RDL Report**, name it **DemoReport**, and click **Add**.
3. In the Solution Explorer, click to select the report, and in the Properties window, change the **Copy to Output Directory** property to **Copy always**.
4. From the ActiveReports 14 RDL Report Toolbox, drag a Table report control onto the report.

> **Note**: In case you are still working on Page report layout, set the FixedSize property of the Table control to display all data on one page.

5. Click inside the table to reveal the table adorners, then right-click the table adorner to the left of the footer row and select **Delete Rows**. The footer row is removed from the table.
6. In the Report Explorer, select each of the textboxes in turn and set the properties as in the following table. (If you do not see the Report Explorer, from the View menu, select Other Windows, then Report Explorer 14.)

TextBox Name	Value Property	BackgroundColor Property
TextBox1	Name	MediumSlateBlue
TextBox2	Role	MediumSlateBlue

TextBox3	City	MediumSlateBlue
TextBox4	=Fields!FirstName.Value & " " & Fields!LastName.Value	
TextBox5	=Fields!Role.Value	
TextBox6	=Fields!City.Value	

7. In the Report Explorer, select the Table1 node and in the Properties window, set the **Location** property to 0in, 1in and the **Size** property to **6in, 0.5in** to make the table wide enough to see all of the data.
8. With Table1 still selected in the Properties window, in the **DataSetName** property, enter the text **DataSet1**.

To add a class library project to the solution to contain the custom data provider

1. From the File menu, select **Add**, then **New Project**.
2. In the Add New Project dialog, select **Class Library**, and name the project **CustomDataProvider**.
3. In the Solution Explorer, right-click the default class and select **Delete**. (We will add our classes to a folder below.)
4. Install packages from **nuget** as follows:
   i) Go to **Tools** > **Nuget Package Manager** > **Manage Nuget Packages for Solution**...
   ii) Browse the following packages one by one and click Install.
   GrapeCity.ActiveReports
   GrapeCity.ActiveReports.Extensibility
5. Right-click the CustomDataProvider project and select **Add**, then **New Folder**, and name the folder **CSVDataProvider**.
6. Right-click the folder and select **Add**, then **Class**, then name the class **CsvColumn** and add code like the following to replace the default stub in the class.

Visual Basic code

Visual Basic code. Paste it to replace the default stub in the class.

```
Namespace CSVDataProvider
        ' Represents information about fields in the data source.
        Friend Structure CsvColumn
                Private ReadOnly _fieldName As String
                Private ReadOnly _dataType As Type

                ' Creates a new instance of the CsvColumn class.
                ' The fieldName parameter is the name of the field represented by
this instance of the CsvColumn.
                ' The dataType parameter is the Type of the field represented by
this instance of the CsvColumn.
                Public Sub New(fieldName As String, dataType As Type)
                        If fieldName Is Nothing Then
                                Throw New ArgumentNullException("fieldName")
                        End If
                        If dataType Is Nothing Then
                                Throw New ArgumentNullException("dataType")
                        End If
                        _fieldName = fieldName
                        _dataType = dataType
                End Sub
```

```
        ' Gets the name of the field represented by this instance of the
CsvColumn.
        Public ReadOnly Property FieldName() As String
            Get
                Return _fieldName
            End Get
        End Property


        ' Gets the the Type of the field represented by this instance of the
CsvColumn.
        Public ReadOnly Property DataType() As Type
            Get
                Return _dataType
            End Get
        End Property


        ' Returns a String that represents this instance of the CsvColumn.
        Public Overrides Function ToString() As String
            Return [String].Concat(New String() {FieldName, "(",
DataType.ToString(), ")"})
        End Function


        ' Determines whether two CsvColumn instances are equal.
        ' The obj represents the CsvColumn to compare with the current
CsvColumn.
        ' Returns True if the specified CsvColumn is equal to the current
CsvColumn; otherwise, False.
        Public Overrides Function Equals(obj As Object) As Boolean
            Dim flag As Boolean

            If TypeOf obj Is CsvColumn Then
                flag = Equals(CType(obj, CsvColumn))
            Else
                flag = False
            End If
            Return flag
        End Function

        Private Overloads Function Equals(column As CsvColumn) As Boolean
            Return column.FieldName = FieldName
        End Function
```

```
                ' Serves as a hash function for a CsvColumn, suitable for use in
hashing algorithms and data structures like a hash table.
                ' Returns a hash code for the current CsvColumn instance.
                Public Overrides Function GetHashCode() As Integer
                        Return (FieldName.GetHashCode() + DataType.GetHashCode())
                End Function
        End Structure
End Namespace
```

C# code

C# code. Paste it to replace the default stub in the class.

```
using System;

namespace CustomDataProvider.CSVDataProvider
{
        // Represents information about fields in the data source.
        internal struct CsvColumn
        {
                private readonly string _fieldName;
                private readonly Type _dataType;

                // Creates a new instance of the CsvColumn class.
                // The fieldName parameter is the name of the field represented by
this instance of the CsvColumn.
                // The dataType parameter is the Type of the field represented by
this instance of the CsvColumn.
                public CsvColumn(string fieldName, Type dataType)
                {
                        if (fieldName == null)
                                throw new ArgumentNullException("fieldName");
                        if (dataType == null)
                                throw new ArgumentNullException("dataType");
                        _fieldName = fieldName;
                        _dataType = dataType;
                }


                // Gets the name of the field represented by this instance of the
CsvColumn.
                public string FieldName
                {
                        get { return _fieldName; }
                }


                // Gets the the Type of the field represented by this instance of
the CsvColumn.
```

```
                public Type DataType
                {
                        get { return _dataType; }
                }


               // Returns a String that represents this instance of the CsvColumn.
                public override string ToString()
                {
                        return String.Concat(new string[] {FieldName, "(",
DataType.ToString(), ")"});
                }


                // Determines whether two CsvColumn instances are equal.
                // The obj represents the CsvColumn to compare with the current
CsvColumn.
                // Returns True if the specified CsvColumn is equal to the current
CsvColumn; otherwise, False.
                public override bool Equals(object obj)
                {
                        bool flag;

                        if (obj is CsvColumn)
                        {
                                flag = Equals((CsvColumn) obj);
                        }
                        else
                        {
                                flag = false;
                        }
                        return flag;
                }

                private bool Equals(CsvColumn column)
                {
                        return column.FieldName == FieldName;
                }


                // Serves as a hash function for a CsvColumn, suitable for use in
hashing algorithms and data structures like a hash table.
               // Returns a hash code for the current CsvColumn instance.
                public override int GetHashCode()
                {
                        return (FieldName.GetHashCode() + DataType.GetHashCode());
                }
        }
```

```
}
```

7. Right-click the CSVDataProvider folder and select **Add**, then **Class**, then name the class **CsvDataReader** and add code like the following to replace the default stub in the class.

Visual Basic code

Visual Basic code. Paste it to replace the default stub in the class.

```
Imports System
Imports System.Collections
Imports System.Globalization
Imports System.IO
Imports System.Text.RegularExpressions
Imports GrapeCity.ActiveReports.Extensibility.Data

Namespace CSVDataProvider

    ' Provides an implementation of IDataReader for the .NET Framework CSV Data
Provider.
    Friend Class CsvDataReader
        Implements IDataReader
        'NOTE: HashcodeProvider and Comparer need to be case-insensitive since
TypeNames are capitalized differently in places.
        'Otherwise data types end up as strings when using Int32 vs int32.
        Private _typeLookup As New
Hashtable(StringComparer.Create(CultureInfo.InvariantCulture, False))

        Private _columnLookup As New Hashtable()
        Private _columns As Object()
        Private _textReader As TextReader
        Private _currentRow As Object()


        'The regular expressions are set to be pre-compiled to make it faster. Since
we were concerned about
        'multi-threading, we made the properties read-only so no one can change any
properties on these objects.
        Private Shared ReadOnly _rxDataRow As New Regex(",(?=(?:[^""]*""[^""]*"")*
(?![^""]*""))", RegexOptions.Compiled)
        'Used to parse the data rows.

        Private Shared ReadOnly _rxHeaderRow As New Regex("(?<fieldName>(\w*\s*)*)\
((?<fieldType>\w*)\)", RegexOptions.Compiled)
        'Used to parse the header rows.


        ' Creates a new instance of the CsvDataReader class.
        ' The textReader parameter represents the TextReader to use to read the
data.
```

```
        Public Sub New(textReader As TextReader)
            _textReader = textReader
            ParseCommandText()
        End Sub


        ' Parses the passed-in command text.
        Private Sub ParseCommandText()
            If _textReader.Peek() = -1 Then
                Return
            End If
            'Command text is empty or at the end already.
            FillTypeLookup()

            Dim header As String = _textReader.ReadLine()
            header = AddDefaultTypeToHeader(header)

            If Not ParseHeader(header) Then
                Throw New InvalidOperationException( _
                "Field names and types are not defined. " & _
                "The first line in the CommandText must contain the field names and
data types. e.g FirstName(string)")
            End If
        End Sub

        'A hashtable is used to return a type for the string value used in the
header text.
        Private Sub FillTypeLookup()
            _typeLookup.Add("string", GetType([String]))
            _typeLookup.Add("byte", GetType([Byte]))
            _typeLookup.Add("boolean", GetType([Boolean]))
            _typeLookup.Add("datetime", GetType(DateTime))
            _typeLookup.Add("decimal", GetType([Decimal]))
            _typeLookup.Add("double", GetType([Double]))
            _typeLookup.Add("int16", GetType(Int16))
            _typeLookup.Add("int32", GetType(Int32))
            _typeLookup.Add("int", GetType(Int32))
            _typeLookup.Add("integer", GetType(Int32))
            _typeLookup.Add("int64", GetType(Int64))
            _typeLookup.Add("sbyte", GetType([SByte]))
            _typeLookup.Add("single", GetType([Single]))
            _typeLookup.Add("time", GetType(DateTime))
            _typeLookup.Add("date", GetType(DateTime))
            _typeLookup.Add("uint16", GetType(UInt16))
            _typeLookup.Add("uint32", GetType(UInt32))
            _typeLookup.Add("uint64", GetType(UInt64))
        End Sub
```

```
        ' Returns a type based on the string value passed in from the header text
string. If no match is found,
        ' a string type is returned.
        ' The fieldType parameter represents the String value from the header
command text string.
        Private Function GetFieldTypeFromString(fieldType As String) As Type
            If _typeLookup.Contains(fieldType) Then
                Return TryCast(_typeLookup(fieldType), Type)
            End If
            Return GetType([String])
        End Function


        ' Parses the first line in the passed-in command text string to create the
field names and field data types.
        ' The field information is stored in a CsvColumn struct, and these column
info items are stored
        ' in an ArrayList. The column name is also added to a hashtable for easy
lookup later.
        ' The header parameter represents the header string that contains all the
fields.
        ' Returns True if it can parse the header string; otherwise False.
        Private Function ParseHeader(header As String) As Boolean
            Dim fieldName As String
            Dim index As Integer = 0
            If header.IndexOf("(") = -1 Then
                Return False
            End If

            Dim matches As MatchCollection = _rxHeaderRow.Matches(header)
            _columns = New Object(matches.Count - 1) {}
            For Each match As Match In matches
                fieldName = match.Groups("fieldName").Value
                Dim fieldType As Type =
GetFieldTypeFromString(match.Groups("fieldType").Value)
                _columns.SetValue(New CsvColumn(fieldName, fieldType), index)
                _columnLookup.Add(fieldName, index)
                index += 1
            Next

            Return True
        End Function


        ' Ensures that the header contains columns in the form of name(type)
        ' The line parameter represents the raw header line from the file to fix up.
        ' Returns a modified header with default types appended to column names.
```

```
        Private Shared Function AddDefaultTypeToHeader(line As String) As String
            Const ColumnWithDataTypeRegex As String = "[""]?\w+[\""]?\(.+\)"
            Dim columns As String() = line.Split(New String() {","},
StringSplitOptions.None)
            Dim ret As String = Nothing
            For Each column As String In columns
                If Not String.IsNullOrEmpty(ret) Then
                    ret += ","
                End If
                If Not Regex.Match(column, ColumnWithDataTypeRegex).Success Then
                    ret += column + "(string)"
                Else
                    ret += column
                End If
            Next
            Return ret
        End Function


        ' Parses a row of data using a regular expression and stores the information
inside an object
        ' array that is the current row of data.
        ' If the row does not have the correct number of fields, an exception is
raised.
        ' The dataRow parameter represents the String value representing a comma
delimited data row.
        ' Returns True if it can parse the data string; otherwise False.
        Private Function ParseDataRow(dataRow As String) As Boolean
            Dim index As Integer = 0
            Dim tempData As String() = _rxDataRow.Split(dataRow)

            _currentRow = New Object(tempData.Length - 1) {}
            If tempData.Length <> _columns.Length Then
                Dim [error] As String = String.Format(CultureInfo.InvariantCulture,
_
                                      "Invalid row ""{0}"". The row does not contain the
same number of data columns as the table header definition.", dataRow)
                Throw New InvalidOperationException([error])
            End If
            For i As Integer = 0 To tempData.Length - 1
                Dim value As String = tempData(i)

                If value.Length > 1 Then
                    If value.IndexOf("""c, 0) = 0 AndAlso value.IndexOf("""c, 1) =
value.Length - 1 Then
                        value = value.Substring(1, value.Length - 2)
                    End If
                End If
```

```
                _currentRow.SetValue(ConvertValue(GetFieldType(index), value),
index)
                index += 1
            Next
            Return True
        End Function


        ' Coverts the string value coming from the command text to the appropriate
data type, based on the field's type.
        ' This also checks a few string value rules to decide if a String.Empty of
System.Data.DBNull needs to be returned.
        ' The type parameter represents the Type of the current column the data
belongs to.
        ' The originalValue parameter represents the String value coming from the
command text.
        ' Returns the object resulting from the converted string, based on the type.
        Private Function ConvertValue(type As Type, originalValue As String) As
Object
            Dim fieldType As Type = type
            Dim invariantCulture As CultureInfo = CultureInfo.InvariantCulture
            Try
                If originalValue = """""" OrElse originalValue = " " Then
                    Return String.Empty
                End If
                If originalValue = "" Then
                    Return DBNull.Value
                End If
                If originalValue = "DBNull" Then
                    Return DBNull.Value
                End If
                If fieldType.Equals(GetType([String])) Then
                    Return originalValue.Trim()
                End If
                If fieldType.Equals(GetType(Int32)) Then
                    Return Convert.ToInt32(originalValue, invariantCulture)
                End If
                If fieldType.Equals(GetType([Boolean])) Then
                    Return Convert.ToBoolean(originalValue, invariantCulture)
                End If
                If fieldType.Equals(GetType(DateTime)) Then
                    Return Convert.ToDateTime(originalValue, invariantCulture)
                End If
                If fieldType.Equals(GetType([Decimal])) Then
                    Return Convert.ToDecimal(originalValue, invariantCulture)
                End If
                If fieldType.Equals(GetType([Double])) Then
                    Return Convert.ToDouble(originalValue, invariantCulture)
```

```
                End If
                If fieldType.Equals(GetType(Int16)) Then
                    Return Convert.ToInt16(originalValue, invariantCulture)
                End If
                If fieldType.Equals(GetType(Int64)) Then
                    Return Convert.ToInt64(originalValue, invariantCulture)
                End If
                If fieldType.Equals(GetType([Single])) Then
                    Return Convert.ToSingle(originalValue, invariantCulture)
                End If
                If fieldType.Equals(GetType([Byte])) Then
                    Return Convert.ToByte(originalValue, invariantCulture)
                End If
                If fieldType.Equals(GetType([SByte])) Then
                    Return Convert.ToSByte(originalValue, invariantCulture)
                End If
                If fieldType.Equals(GetType(UInt16)) Then
                    Return Convert.ToUInt16(originalValue, invariantCulture)
                End If
                If fieldType.Equals(GetType(UInt32)) Then
                    Return Convert.ToUInt32(originalValue, invariantCulture)
                End If
                If fieldType.Equals(GetType(UInt64)) Then
                    Return Convert.ToUInt64(originalValue, invariantCulture)
                End If
            Catch e As Exception
                Throw New InvalidOperationException(String.Format("Input value '{0}'
could not be converted to the type '{1}'.", originalValue, type), e)
            End Try
            'If no match is found return DBNull instead.
            Return DBNull.Value
        End Function

#Region "IDataReader Members"


       ' Advances the CsvDataReader to the next record.
        ' Returns True if there are more rows; otherwise, False.
        Public Function Read() As Boolean Implements IDataReader.Read
            If _textReader.Peek() > -1 Then
                ParseDataRow(_textReader.ReadLine())
            Else
                Return False
            End If

            Return True
        End Function
```

```
#End Region

#Region "IDisposable Members"


        ' Releases the resources used by the CsvDataReader.
        Public Sub Dispose() Implements IDisposable.Dispose
            Dispose(True)
            GC.SuppressFinalize(Me)
        End Sub

        Private Sub Dispose(disposing As Boolean)
            If disposing Then
                If _textReader IsNot Nothing Then
                    _textReader.Close()
                End If
            End If

            _typeLookup = Nothing
            _columnLookup = Nothing
            _columns = Nothing
            _currentRow = Nothing
        End Sub


        ' Allows an Object to attempt to free resources and perform
        ' other cleanup operations before the Object is reclaimed by garbage
collection.
        Protected Overrides Sub Finalize()
            Try
                Dispose(False)
            Finally
                MyBase.Finalize()
            End Try
        End Sub

#End Region

#Region "IDataRecord Members"


        ' Gets the number of columns in the current row.
        Public ReadOnly Property FieldCount() As Integer Implements
IDataRecord.FieldCount
            Get
                Return _columns.Length
            End Get
        End Property
```

```
        ' The i parameter represents the index of the field to find.
        ' Returns the Type information corresponding to the type of Object that
would be returned from GetValue.
        Public Function GetFieldType(i As Integer) As Type Implements
IDataReader.GetFieldType
            If i > _columns.Length - 1 Then
                Return Nothing
            End If

            Return DirectCast(_columns.GetValue(i), CsvColumn).DataType
        End Function


        ' Gets the name for the field to find.
        ' The i parameter represents the index of the field to find.
        ' Returns the name of the field or an empty string (""), if there is no
value to return.
        Public Function GetName(i As Integer) As String Implements
IDataRecord.GetName
            If i > _columns.Length - 1 Then
                Return String.Empty
            End If


            Return DirectCast(_columns.GetValue(i), CsvColumn).FieldName
        End Function


        ' The name parameter represents the name of the field to find.
        ' Returns the index of the named field.
        Public Function GetOrdinal(name As String) As Integer Implements
IDataRecord.GetOrdinal
            Dim value As Object = _columnLookup(name)
            If value Is Nothing Then
                Throw New IndexOutOfRangeException("name")
            End If
            Return CInt(value)
        End Function


        ' The i parameter represents the index of the field to find.
        ' Returns the Object which contains the value of the specified field.
        Public Function GetValue(i As Integer) As Object Implements
IDataRecord.GetValue
            If i > _columns.Length - 1 Then
                Return Nothing
```

```
            End If

            Return _currentRow.GetValue(i)
        End Function

        Public Overridable Function GetData(fieldIndex As Integer) As IDataReader
Implements IDataReader.GetData
            Throw New NotSupportedException()
        End Function

#End Region
    End Class
End Namespace
```

C# code

C# code. Paste it to replace the default stub in the class.

```
using System;
using System.Collections;
using System.Globalization;
using System.IO;
using System.Text.RegularExpressions;
using GrapeCity.ActiveReports.Extensibility.Data;

namespace CustomDataProvider.CSVDataProvider
{

        // Provides an implementation of IDataReader for the .NET Framework CSV Data
Provider.
        internal class CsvDataReader : IDataReader
        {
                //NOTE: HashcodeProvider and Comparer need to be case-insensitive
since TypeNames are capitalized differently in places.
                                //Otherwise data types end up as strings when using
Int32 vs int32.
                private Hashtable _typeLookup =
                        new
Hashtable(StringComparer.Create(CultureInfo.InvariantCulture, false));
                private Hashtable _columnLookup = new Hashtable();
                private object[] _columns;
                private TextReader _textReader;
                private object[] _currentRow;


                //The regular expressions are set to be pre-compiled to make it
faster. Since we were concerned about
                //multi-threading, we made the properties read-only so no one can
change any properties on these objects.
```

```
                private static readonly Regex _rxDataRow = new Regex(@",(?=(?:
[^""]*""[^""]*"")*(?![^""]*""))", RegexOptions.Compiled);
                        //Used to parse the data rows.

                private static readonly Regex _rxHeaderRow =
                        new Regex(@"(?<fieldName>(\w*\s*)*)\((?<fieldType>\w*)\)",
RegexOptions.Compiled);
                        //Used to parse the header rows.


                // Creates a new instance of the CsvDataReader class.
                // The textReader parameter represents the TextReader to use to read
the data.
                public CsvDataReader(TextReader textReader)
                {
                        _textReader = textReader;
                        ParseCommandText();
                }


                // Parses the passed-in command text.
                private void ParseCommandText()
                {
                        if (_textReader.Peek() == -1)
                                return; //Command text is empty or at the end
already.

                        FillTypeLookup();

                        string header = _textReader.ReadLine();
                        header = AddDefaultTypeToHeader(header);

                        if (!ParseHeader(header))
                                throw new InvalidOperationException(
                                        "Field names and types are not defined. The
first line in the CommandText must contain the field names and data types. e.g
FirstName(string)");
                }

                //A hashtable is used to return a type for the string value used in
the header text.
                private void FillTypeLookup()
                {
                        _typeLookup.Add("string", typeof (String));
                        _typeLookup.Add("byte", typeof (Byte));
                        _typeLookup.Add("boolean", typeof (Boolean));
                        _typeLookup.Add("datetime", typeof (DateTime));
                        _typeLookup.Add("decimal", typeof (Decimal));
```

```
                        _typeLookup.Add("double", typeof (Double));
                        _typeLookup.Add("int16", typeof (Int16));
                        _typeLookup.Add("int32", typeof (Int32));
                        _typeLookup.Add("int", typeof (Int32));
                        _typeLookup.Add("integer", typeof (Int32));
                        _typeLookup.Add("int64", typeof (Int64));
                        _typeLookup.Add("sbyte", typeof (SByte));
                        _typeLookup.Add("single", typeof (Single));
                        _typeLookup.Add("time", typeof (DateTime));
                        _typeLookup.Add("date", typeof (DateTime));
                        _typeLookup.Add("uint16", typeof (UInt16));
                        _typeLookup.Add("uint32", typeof (UInt32));
                        _typeLookup.Add("uint64", typeof (UInt64));
                }


                // Returns a type based on the string value passed in from the
header text string. If no match is found, a string type is returned.
                // The fieldType parameter represents the String value from the
header command text string.
                private Type GetFieldTypeFromString(string fieldType)
                {
                        if (_typeLookup.Contains(fieldType))
                                return _typeLookup[fieldType] as Type;
                        return typeof (String);
                }


                // Parses the first line in the passed-in command text string to
create the field names and field data types. The field information
                // is stored in a CsvColumn struct, and these column info items are
stored in an ArrayList. The column name is also added
                // to a hashtable for easy lookup later.

                // The header parameter represents the header string that contains
all the fields.
                // Returns True if it can parse the header string; otherwise False.
                private bool ParseHeader(string header)
                {
                        string fieldName;
                        int index = 0;
                        if (header.IndexOf("(") == -1)
                                return false;

                        MatchCollection matches = _rxHeaderRow.Matches(header);
                        _columns = new object[matches.Count];
                        foreach (Match match in matches)
                        {
```

```
                                fieldName = match.Groups["fieldName"].Value;
                                Type fieldType =
GetFieldTypeFromString(match.Groups["fieldType"].Value);
                                _columns.SetValue(new CsvColumn(fieldName,
fieldType), index);
                                _columnLookup.Add(fieldName, index);
                                index++;
                        }

                        return true;
                }


                // Ensures that the header contains columns in the form of
name(type)
                // The line parameter represents the raw header line from the file
to fix up.
                // Returns a modified header with default types appended to column
names.
                private static string AddDefaultTypeToHeader(string line)
                {
                        const string ColumnWithDataTypeRegex = @"[""]?\w+[\""]?\
(.+\)";
                        string[] columns = line.Split(new string[] { "," },
StringSplitOptions.None);
                        string ret = null;
                        foreach (string column in columns)
                        {
                                if (!string.IsNullOrEmpty(ret))
                                        ret += ",";
                                if (!Regex.Match(column,
ColumnWithDataTypeRegex).Success)
                                {
                                        ret += column + "(string)";
                                }
                                else
                                {
                                        ret += column;
                                }
                        }
                        return ret;
                }


                // Parses a row of data using a regular expression and stores the
information inside an object array that is the current row of data.
                // If the row does not have the correct number of fields, an
exception is raised.
```

```
                // The dataRow parameter represents the String value representing a
comma delimited data row.
                // Returns True if it can parse the data string; otherwise False.
                private bool ParseDataRow(string dataRow)
                {
                        int index = 0;
                        string[] tempData = _rxDataRow.Split(dataRow);

                        _currentRow = new object[tempData.Length];
                        if (tempData.Length != _columns.Length)
                        {
                                string error =
                                        string.Format(CultureInfo.InvariantCulture,
                                                "Invalid row \"{0}\". The row
does not contain the same number of data columns as the table header definition.",
                                                dataRow);
                                throw new InvalidOperationException(error);
                        }
                        for (int i = 0; i < tempData.Length; i++)
                        {
                                string value = tempData[i];

                                if (value.Length > 1)
                                {
                                        if (value.IndexOf('"', 0) == 0 &&
value.IndexOf('"', 1) == value.Length - 1)
                                                value = value.Substring(1,
value.Length - 2);
                                }

_currentRow.SetValue(ConvertValue(GetFieldType(index), value), index);
                                index++;
                        }
                        return true;
                }


                // Coverts the string value coming from the command text to the
appropriate data type, based on the field's type.
                // This also checks a few string value rules to decide if a
String.Empty of System.Data.DBNull needs to be returned.
                // The type parameter represents the Type of the current column the
data belongs to.
                // The originalValue parameter represents the String value coming
from the command text.
                // Returns the object resulting from the converted string, based on
the type.
                private object ConvertValue(Type type, string originalValue)
```

```
                {
                        Type fieldType = type;
                        CultureInfo invariantCulture = CultureInfo.InvariantCulture;
                        try
                        {
                                if (originalValue == "\"\"" || originalValue == " ")
                                        return string.Empty;
                                if (originalValue == "")
                                        return DBNull.Value;
                                if (originalValue == "DBNull")
                                        return DBNull.Value;
                                if (fieldType.Equals(typeof (String)))
                                        return originalValue.Trim();
                                if (fieldType.Equals(typeof (Int32)))
                                        return Convert.ToInt32(originalValue,
invariantCulture);
                                if (fieldType.Equals(typeof (Boolean)))
                                        return Convert.ToBoolean(originalValue,
invariantCulture);
                                if (fieldType.Equals(typeof (DateTime)))
                                        return Convert.ToDateTime(originalValue,
invariantCulture);
                                if (fieldType.Equals(typeof (Decimal)))
                                        return Convert.ToDecimal(originalValue,
invariantCulture);
                                if (fieldType.Equals(typeof (Double)))
                                        return Convert.ToDouble(originalValue,
invariantCulture);
                                if (fieldType.Equals(typeof (Int16)))
                                        return Convert.ToInt16(originalValue,
invariantCulture);
                                if (fieldType.Equals(typeof (Int64)))
                                        return Convert.ToInt64(originalValue,
invariantCulture);
                                if (fieldType.Equals(typeof (Single)))
                                        return Convert.ToSingle(originalValue,
invariantCulture);
                                if (fieldType.Equals(typeof (Byte)))
                                        return Convert.ToByte(originalValue,
invariantCulture);
                                if (fieldType.Equals(typeof (SByte)))
                                        return Convert.ToSByte(originalValue,
invariantCulture);
                                if (fieldType.Equals(typeof (UInt16)))
                                        return Convert.ToUInt16(originalValue,
invariantCulture);
                                if (fieldType.Equals(typeof (UInt32)))
                                        return Convert.ToUInt32(originalValue,
```

```
invariantCulture);
                                if (fieldType.Equals(typeof (UInt64)))
                                        return Convert.ToUInt64(originalValue,
invariantCulture);
                        }
                        catch (Exception e)
                        {
                                throw new InvalidOperationException(
                                        string.Format("Input value '{0}' could not
be converted to the type '{1}'.", originalValue, type), e);
                        }
                        //If no match is found return DBNull instead.
                        return DBNull.Value;
                }

                #region IDataReader Members


                // Advances the CsvDataReader to the next record.
                // Returns True if there are more rows; otherwise, False.
                public bool Read()
                {
                        if (_textReader.Peek() > -1)
                                ParseDataRow(_textReader.ReadLine());
                        else
                                return false;

                        return true;
                }

                #endregion

                #region IDisposable Members


                // Releases the resources used by the CsvDataReader.
                public void Dispose()
                {
                        Dispose(true);
                        GC.SuppressFinalize(this);
                }

                private void Dispose(bool disposing)
                {
                        if (disposing)
                        {
                                if (_textReader != null)
                                        _textReader.Close();
```

```
                        }

                        _typeLookup = null;
                        _columnLookup = null;
                        _columns = null;
                        _currentRow = null;
                }


                // Allows an Object to attempt to free resources and perform other
cleanup operations before the Object is reclaimed by garbage collection.
                ~CsvDataReader()
                {
                        Dispose(false);
                }

                #endregion

                #region IDataRecord Members


                // Gets the number of columns in the current row.
                public int FieldCount
                {
                        get { return _columns.Length; }
                }


                // The i parameter represents the index of the field to find.
                // Returns the Type information corresponding to the type of Object
that would be returned from GetValue.
                public Type GetFieldType(int i)
                {
                        if (i > _columns.Length - 1)
                                return null;

                        return ((CsvColumn) _columns.GetValue(i)).DataType;
                }


                // Gets the name for the field to find.
                // The i parameter represents the index of the field to find.
                // Returns the name of the field or an empty string (""), if there
is no value to return.
                public string GetName(int i)
                {
                        if (i > _columns.Length - 1)
                                return string.Empty;
```

```
                return ((CsvColumn) _columns.GetValue(i)).FieldName;
            }


            // The name parameter represents the name of the field to find.
            // Returns the index of the named field.
            public int GetOrdinal(string name)
            {
                object value = _columnLookup[name];
                if (value == null)
                    throw new IndexOutOfRangeException("name");
                return (int) value;
            }


            // The i parameter represents the index of the field to find.
            // Returns the Object which contains the value of the specified
field.
            public object GetValue(int i)
            {
                if (i > _columns.Length - 1)
                    return null;

                return _currentRow.GetValue(i);
            }

            public virtual IDataReader GetData(int fieldIndex)
            {
                throw new NotSupportedException();
            }

            #endregion
        }
}
```

8. Right-click the CSVDataProvider folder and select **Add**, then **Class**, then name the class **CsvCommand** and add code like the following to replace the default stub in the class.

Visual Basic code

Visual Basic code. Paste it to replace the default stub in the class.

```
Imports System
Imports System.IO
Imports GrapeCity.ActiveReports.Extensibility.Data

Namespace CSVDataProvider
```

```
    ' Provides the IDbCommand implementation for the .NET Framework CSV Data
Provider.
    Public NotInheritable Class CsvCommand
        Implements IDbCommand
        Private _commandText As String
        Private _connection As IDbConnection
        Private _commandTimeout As Integer
        Private _commandType As CommandType


        ' Creates a new instance of the CsvCommand class.
        Public Sub New()
            Me.New(String.Empty)
        End Sub


        ' Creates a new instance of the CsvCommand class with command text.
        ' The commandText parameter represents the command text.
        Public Sub New(commandText As String)
            Me.New(commandText, Nothing)
        End Sub


        ' Creates a new instance of the CsvCommand class with command text and a
CsvConnection.
        ' The commandText parameter represents the command text.
        ' The connection parameter represents a CsvConnection to a data source.
        Public Sub New(commandText As String, connection As CsvConnection)
            _commandText = commandText
            _connection = connection
        End Sub


        ' Gets or sets the command to execute at the data source.
        Public Property CommandText() As String Implements IDbCommand.CommandText
            Get
                Return _commandText
            End Get
            Set(value As String)
                _commandText = value
            End Set
        End Property


        ' Gets or sets the wait time before terminating an attempt to execute the
command and generating an error.
        Public Property CommandTimeout() As Integer Implements
IDbCommand.CommandTimeout
```

```
            Get
                Return _commandTimeout
            End Get

            Set(value As Integer)
                _commandTimeout = value
            End Set
        End Property


        ' Gets or sets a value indicating how the CommandText property is
interpreted.
        ' Remarks: We don't use this one for the Csv Data Provider.
        Public Property CommandType() As CommandType Implements
IDbCommand.CommandType
            Get
                Return _commandType
            End Get

            Set(value As CommandType)
                _commandType = value
            End Set
        End Property


        ' Gets or sets the CsvConnection used by this instance of the CsvCommand.
        Public Property Connection() As IDbConnection
            Get
                Return _connection
            End Get

            Set(value As IDbConnection)
                _connection = value
            End Set
        End Property


       ' Sends the CommandText to the CsvConnection, and builds a CsvDataReader
using one of the CommandBehavior values.
        ' The behavior parameter represents a CommandBehavior value.
        ' Returns a CsvDataReader object.
        Public Function ExecuteReader(behavior As CommandBehavior) As IDataReader
Implements IDbCommand.ExecuteReader
            Return New CsvDataReader(New StringReader(_commandText))
        End Function


        ' Returns a string that represents the command text with the parameters
```

```
expanded into constants.
        Public Function GenerateRewrittenCommandText() As String Implements
IDbCommand.GenerateRewrittenCommandText
            Return _commandText
        End Function


        ' Sends the CommandText to the CsvConnection and builds a CsvDataReader.
        ' Returns a CsvDataReader object.
        Public Function ExecuteReader() As IDataReader Implements
IDbCommand.ExecuteReader
            Return ExecuteReader(CommandBehavior.SchemaOnly)
        End Function

#Region "Non implemented IDbCommand Members"

        Public ReadOnly Property Parameters() As IDataParameterCollection Implements
IDbCommand.Parameters
            Get
                Throw New NotImplementedException()
            End Get
        End Property

        Public Property Transaction() As IDbTransaction Implements
IDbCommand.Transaction
            Get
                Throw New NotImplementedException()
            End Get

            Set(value As IDbTransaction)
                Throw New NotImplementedException()
            End Set
        End Property

        Public Sub Cancel() Implements IDbCommand.Cancel

        End Sub

        Public Function CreateParameter() As IDataParameter Implements
IDbCommand.CreateParameter
            Throw New NotImplementedException()
        End Function

#End Region

#Region "IDisposable Members"
```

```
        ' Releases the resources used by the CsvCommand.
        Public Sub Dispose() Implements IDisposable.Dispose
            Dispose(True)
            GC.SuppressFinalize(Me)
        End Sub


        Private Sub Dispose(disposing As Boolean)
            If disposing Then
                If _connection IsNot Nothing Then
                    _connection.Dispose()
                    _connection = Nothing
                End If
            End If
        End Sub

#End Region
    End Class
End Namespace
```

C# code

C# code. Paste it to replace the default stub in the class.

```
using System;
using System.IO;
using GrapeCity.ActiveReports.Extensibility.Data;

namespace CustomDataProvider.CSVDataProvider
{

       // Provides the IDbCommand implementation for the .NET Framework CSV Data
Provider.
        public sealed class CsvCommand : IDbCommand
        {
                private string _commandText;
                private IDbConnection _connection;
                private int _commandTimeout;
                private CommandType _commandType;


                /// Creates a new instance of the CsvCommand class.
                public CsvCommand()
                        : this(string.Empty)
                {
                }


                // Creates a new instance of the CsvCommand class with command text.
```

```
		// The commandText parameter represents the command text.
		public CsvCommand(string commandText)
			: this(commandText, null)
		{
		}


		// Creates a new instance of the CsvCommand class with command text
and a CsvConnection.
		// The commandText parameter represents the command text.
		// The connection parameter represents a CsvConnection to a data
source.?
		public CsvCommand(string commandText, CsvConnection connection)
		{
			_commandText = commandText;
			_connection = connection;
		}


		// Gets or sets the command to execute at the data source.
		public string CommandText
		{
			get { return _commandText; }
			set { _commandText = value; }
		}


		// Gets or sets the wait time before terminating an attempt to
execute the command and generating an error.
		public int CommandTimeout
		{
			get { return _commandTimeout; }

			set { _commandTimeout = value; }
		}


		// Gets or sets a value indicating how the CommandText property is
interpreted.
		// Remarks: We don't use this one for the Csv Data Provider.
		public CommandType CommandType
		{
			get { return _commandType; }

			set { _commandType = value; }
		}
```

```
                // Gets or sets the CsvConnection used by this instance of the
CsvCommand.
                public IDbConnection Connection
                {
                        get { return _connection; }

                        set { _connection = value; }
                }


                // Sends the CommandText to the CsvConnection, and builds a
CsvDataReader using one of the CommandBehavior values.
                // The behavior parameter represents a CommandBehavior value.
                // Returns a CsvDataReader object.
                public IDataReader ExecuteReader(CommandBehavior behavior)
                {
                        return new CsvDataReader(new StringReader(_commandText));
                }


                // Returns a string that represents the command text with the
parameters expanded into constants.
                public string GenerateRewrittenCommandText()
                {
                        return _commandText;
                }


                // Sends the CommandText to the CsvConnection and builds a
CsvDataReader.
                // Returns a CsvDataReader object.
                public IDataReader ExecuteReader()
                {
                        return ExecuteReader(CommandBehavior.SchemaOnly);
                }

                #region Non implemented IDbCommand Members

                public IDataParameterCollection Parameters
                {
                        get { throw new NotImplementedException(); }
                }

                public IDbTransaction Transaction
                {
                        get { throw new NotImplementedException(); }

                        set { throw new NotImplementedException(); }
```

```
                }

                public void Cancel()
                {

                }

                public IDataParameter CreateParameter()
                {
                        throw new NotImplementedException();
                }

                #endregion

                #region IDisposable Members


                // Releases the resources used by the CsvCommand.
                public void Dispose()
                {
                        Dispose(true);
                        GC.SuppressFinalize(this);
                }


                private void Dispose(bool disposing)
                {
                        if (disposing)
                        {
                                if (_connection != null)
                                {
                                        _connection.Dispose();
                                        _connection = null;
                                }
                        }
                }

                #endregion
        }
}
```

9. Right-click the CSVDataProvider folder and select **Add**, then **Class**, then name the class **CsvConnection** and add code like the following to replace the default stub in the class. (You can safely ignore the errors, as they will go away when you add the CsvConnection class.)

Visual Basic code

Visual Basic code. Paste it to replace the default stub in the class.
`Imports System`

```
Imports System.Collections.Specialized
Imports GrapeCity.ActiveReports.Extensibility.Data

Namespace CSVDataProvider

    ' Provides an implementation of IDbConnection for the .NET Framework CSV Data
Provider.
    Public NotInheritable Class CsvConnection
        Implements IDbConnection
        Private _localizedName As String

        ' Creates a new instance of the CsvConnection class.
        Public Sub New()
            _localizedName = "Csv"
        End Sub

       ' Creates a new instance of the CsvConnection class.
       ' The localizedName parameter represents the localized name for the
CsvConnection instance.
        Public Sub New(localizeName As String)
            _localizedName = localizeName
        End Sub

#Region "IDbConnection Members"

        ' Gets or sets the string used to open the connection to the data source.
        ' Remarks: We don't use this one for the Csv Data Provider.
        Public Property ConnectionString() As String Implements
IDbConnection.ConnectionString
            Get
                Return String.Empty
            End Get

            Set(value As String)


            End Set
        End Property

        ' Gets the amount of time to wait while trying to establish a connection
before terminating
        ' the attempt and generating an error.
        ' Remarks: We don't use this one for the Csv Data Provider.
        Public ReadOnly Property ConnectionTimeout() As Integer Implements
IDbConnection.ConnectionTimeout
            Get
                Throw New NotImplementedException()
            End Get
```

```
        End Property

        ' Begins a data source transaction.
        ' Returns an object representing the new transaction.
        ' Remarks: We don't use this one for the Csv Data Provider.
        Public Function BeginTransaction() As IDbTransaction Implements
IDbConnection.BeginTransaction
            Return Nothing
        End Function

        ' Opens a data source connection.
        ' Remarks: We don't use this one for the Csv Data Provider.
        Public Sub Open() Implements IDbConnection.Open


        End Sub

        ' Closes the connection to the data source. This is the preferred method of
closing any open connection.
        Public Sub Close() Implements IDbConnection.Close
            Dispose()
        End Sub

        ' Creates and returns a CsvCommand object associated with the CsvConnection.
        Public Function CreateCommand() As IDbCommand Implements
IDbConnection.CreateCommand
            Return New CsvCommand(String.Empty)
        End Function

        Public Property DataProviderService() As IDataProviderService Implements
IDbConnection.DataProviderService
            Get
                Return Nothing
            End Get
            Set(value As IDataProviderService)
            End Set
        End Property

#End Region

#Region "IDisposable Members"

        ' Releases the resources used by the CsvConnection.
        Public Sub Dispose() Implements IDisposable.Dispose
            Dispose(True)
            GC.SuppressFinalize(Me)
        End Sub
```

```
        Private Sub Dispose(disposing As Boolean)
        End Sub

       ' Allows an Object to attempt to free resources and perform other cleanup
operations
        ' before the Object is reclaimed by garbage collection.
        Protected Overrides Sub Finalize()
            Try
                Dispose(False)
            Finally
                MyBase.Finalize()
            End Try
        End Sub

#End Region

#Region "IExtension Members"

       ' Gets the localized name of the CsvConnection.
        Public ReadOnly Property LocalizedName() As String Implements
IDbConnection.LocalizedName
            Get
                Return _localizedName
            End Get
        End Property

       ' Specifies any configuration information for this extension.
       ' The configurationSettings parameter represents a NameValueCollection of the
settings.
        Public Sub SetConfiguration(configurationSettings As NameValueCollection)
Implements IDbConnection.SetConfiguration
        End Sub

#End Region
    End Class
End Namespace
```

C# code

C# code. Paste it to replace the default stub in the class.

```
using System;
using System.Collections.Specialized;
using GrapeCity.ActiveReports.Extensibility.Data;

namespace CustomDataProvider.CSVDataProvider
{

        // Provides an implementation of IDbConnection for the .NET Framework CSV
```

```
Data Provider.
        public sealed class CsvConnection : IDbConnection
        {
                private string _localizedName;


                // Creates a new instance of the CsvConnection class.
                public CsvConnection()
                {
                        _localizedName = "Csv";
                }


                // Creates a new instance of the CsvConnection class.
                // The localizedName parameter represents the localized name for the
CsvConnection instance.
                public CsvConnection(string localizeName)
                {
                        _localizedName = localizeName;
                }

                #region IDbConnection Members


               // Gets or sets the string used to open the connection to the data
source.
               // Remarks: We don't use this one for the Csv Data Provider.
                public string ConnectionString
                {
                        get { return string.Empty; }

                        set { ; }
                }


                // Gets the amount of time to wait while trying to establish a
connection before terminating the attempt and generating an error.
                // Remarks: We don't use this one for the Csv Data Provider.
                public int ConnectionTimeout
                {
                        get { throw new NotImplementedException(); }
                }


               // Begins a data source transaction.
               // Returns an object representing the new transaction.
               // Remarks: We don't use this one for the Csv Data Provider.
                public IDbTransaction BeginTransaction()
```

```
                {
                        return null;
                }


               // Opens a data source connection.
               // Remarks: We don't use this one for the Csv Data Provider.
                public void Open()
                {
                        ;
                }


                // Closes the connection to the data source. This is the preferred
method of closing any open connection.
                public void Close()
                {
                        Dispose();
                }


               // Creates and returns a CsvCommand object associated with the
CsvConnection.
                public IDbCommand CreateCommand()
                {
                        return new CsvCommand(string.Empty);
                }

                public IDataProviderService DataProviderService
                {
                        get { return null; }
                        set { }
                }

                #endregion

                #region IDisposable Members


               // Releases the resources used by the CsvConnection.
                public void Dispose()
                {
                        Dispose(true);
                        GC.SuppressFinalize(this);
                }

                private void Dispose(bool disposing)
                {
```

```
                }


                // Allows an Object to attempt to free resources and perform other
cleanup operations before the Object is reclaimed by garbage collection.
                ~CsvConnection()
                {
                        Dispose(false);
                }

                #endregion

                #region IExtension Members


                // Gets the localized name of the CsvConnection.
                public string LocalizedName
                {
                        get { return _localizedName; }
                }


                // Specifies any configuration information for this extension.
                // The configurationSettings parameter represents a
NameValueCollection of the settings.
                public void SetConfiguration(NameValueCollection
configurationSettings)
                {
                }

                #endregion
        }
}
```

10. Right-click the CSVDataProvider folder and select **Add**, then **Class**, then name the class **CsvDataProviderFactory** and add code like the following to replace the default stub in the class.

Visual Basic code

Visual Basic code. Paste it to replace the default stub in the class.

```
Imports GrapeCity.ActiveReports.Extensibility.Data
Imports GrapeCity.BI.Data.DataProviders

Namespace CSVDataProvider

       ' Implements the DataProviderFactory for .NET Framework CSV Data Provider.
        Public Class CsvDataProviderFactory
                Inherits DataProviderFactory


```

```
                ' Creates new instance of the CsvDataProviderFactory class.
                Public Sub New()
                End Sub

               ' Returns a new instance of the the CsvCommand.
                Public Overrides Function CreateCommand() As IDbCommand
                        Return New CsvCommand()
                End Function

                ' Returns a new instance of the the CsvConnection.
                Public Overrides Function CreateConnection() As IDbConnection
                        Return New CsvConnection()
                End Function
        End Class
End Namespace
```

C# code

C# code. Paste it to replace the default stub in the class.

```
using GrapeCity.ActiveReports.Extensibility.Data;
using GrapeCity.BI.Data.DataProviders;

namespace CustomDataProvider.CSVDataProvider
{

       // Implements the DataProviderFactory for .NET Framework CSV Data Provider.
        public class CsvDataProviderFactory : DataProviderFactory
        {

                // Creates new instance of the CsvDataProviderFactory class.
                public CsvDataProviderFactory()
                {
                }

               // Returns a new instance of the the CsvCommand.
                public override IDbCommand CreateCommand()
                {
                        return new CsvCommand();
                }

               // Returns a new instance of the the CsvConnection.
                public override IDbConnection CreateConnection()
                {
                        return new CsvConnection();
                }
        }
}
```

To add a button to the query editor

1. In the Solution Explorer, right-click the CSVDataProvider folder and select **Add**, then **Class**, then name the class **QueryEditor** and add code like the following to replace the default stub in the class.

   Visual Basic code

Visual Basic code. Paste it to replace the default stub in the class.

```
Imports System.Collections.Generic
Imports System.Drawing.Design
Imports System.IO
Imports System.Linq
Imports System.Text
Imports System.Text.RegularExpressions
Imports System.Windows.Forms
Imports System.Windows.Forms.Design


Namespace CustomDataProvider.CSVDataProvider
    Public NotInheritable Class QueryEditor
        Inherits UITypeEditor
        Public Overrides Function GetEditStyle(context As
System.ComponentModel.ITypeDescriptorContext) As UITypeEditorEditStyle
            Return UITypeEditorEditStyle.DropDown
        End Function
        Public Overrides Function EditValue(context As
System.ComponentModel.ITypeDescriptorContext, provider As System.IServiceProvider,
value As Object) As Object
            Dim edSvc As IWindowsFormsEditorService =
DirectCast(provider.GetService(GetType(IWindowsFormsEditorService)),
IWindowsFormsEditorService)
            Dim path = ""
            Dim btn = New Button()
            btn.Text = "Select CSV File..."
            Dim pdg = btn.Padding
            pdg.Bottom += 2
            btn.Padding = pdg
            btn.Click += Sub() Using openDlg = New OpenFileDialog()
                openDlg.Filter = "CSV Files (*.csv)|*.csv|All Files (*.*)|*.*"
                If openDlg.ShowDialog() <> DialogResult.OK Then
                    path = ""
                Else
                    path = openDlg.FileName
                End If
            End Using
            edSvc.DropDownControl(btn)
            If String.IsNullOrEmpty(path) Then
                Return String.Empty
            End If
```

```
            If Not File.Exists(path) Then
                Return String.Empty
            End If
            Return GetCSVQuery(path)
        End Function
        Private Function GetCSVQuery(path As String) As Object
            Dim sr As StreamReader = Nothing
            Try
                sr = New StreamReader(path)
                Dim ret As String = String.Empty
                Dim currentLine As String
                Dim line As Integer = 0
                While (InlineAssignHelper(currentLine, sr.ReadLine())) IsNot Nothing
                    If line = 0 Then
                        ret += ProcessColumnsDefinition(currentLine) &
Convert.ToString(vbCr & vbLf)
                    Else
                        ret += currentLine & Convert.ToString(vbCr & vbLf)
                    End If
                    line += 1
                End While
                Return ret
            Catch generatedExceptionName As IOException
                Return String.Empty
            Finally
                If sr IsNot Nothing Then
                    sr.Close()
                End If
            End Try
        End Function
        Private Function ProcessColumnsDefinition(currentLine As String) As String
            Const  ColumnWithDataTypeRegex As String = "[""]?\w+[\""]?\(.+\)"
            Dim columns As String() = currentLine.Split(New String() {","},
StringSplitOptions.None)
            Dim ret As String = Nothing
            For Each column As String In columns
                If Not String.IsNullOrEmpty(ret) Then
                    ret += ","
                End If
                If Not Regex.Match(column, ColumnWithDataTypeRegex).Success Then
                    ret += column & Convert.ToString("(string)")
                Else
                    ret += column
                End If
            Next
            Return ret
        End Function
        Private Shared Function InlineAssignHelper(Of T)(ByRef target As T, value As
```

```
T) As T
            target = value
            Return value
        End Function
    End Class
End Namespace
```

C# code

C# code. Paste it to replace the default stub in the class.

```
using System;
using System.Collections.Generic;
using System.Drawing.Design;
using System.IO;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Windows.Forms;
using System.Windows.Forms.Design;
namespace CustomDataProvider.CSVDataProvider
{
    public sealed class QueryEditor : UITypeEditor
    {
        public override UITypeEditorEditStyle
GetEditStyle(System.ComponentModel.ITypeDescriptorContext context)
        {
            return UITypeEditorEditStyle.DropDown;
        }
        public override object
EditValue(System.ComponentModel.ITypeDescriptorContext context,
System.IServiceProvider provider, object value)
        {
            IWindowsFormsEditorService edSvc =
(IWindowsFormsEditorService)provider.GetService(typeof(IWindowsFormsEditorService));
            var path = "";
            var btn = new Button();
            btn.Text = "Select CSV File...";
            var pdg = btn.Padding;
            pdg.Bottom += 2;
            btn.Padding = pdg;
            btn.Click += delegate
            {
                using (var openDlg = new OpenFileDialog())
                {
                    openDlg.Filter = "CSV Files (*.csv)|*.csv|All Files (*.*)|*.*";
                    if (openDlg.ShowDialog() != DialogResult.OK)
                        path = "";
                    else
```

```
                    path = openDlg.FileName;
                }
            };
            edSvc.DropDownControl(btn);
            if (string.IsNullOrEmpty(path)) return string.Empty;
            if (!File.Exists(path)) return string.Empty;
            return GetCSVQuery(path);
        }
        private object GetCSVQuery(string path)
        {
            StreamReader sr = null;
            try
            {
                sr = new StreamReader(path);
                string ret = string.Empty;
                string currentLine;
                int line = 0;
                while ((currentLine = sr.ReadLine()) != null)
                {
                    if (line == 0)
                        ret += ProcessColumnsDefinition(currentLine) + "\r\n";
                    else
                        ret += currentLine + "\r\n";
                    line++;
                }
                return ret;
            }
            catch (IOException)
            {
                return string.Empty;
            }
            finally
            {
                if (sr != null)
                    sr.Close();
            }
        }
        private string ProcessColumnsDefinition(string currentLine)
        {
            const string ColumnWithDataTypeRegex = @"[""]?\w+[\""]?\(.+\)";
            string[] columns = currentLine.Split(new string[] { "," },
StringSplitOptions.None);
            string ret = null;
            foreach (string column in columns)
            {
                if (!string.IsNullOrEmpty(ret))
                    ret += ",";
                if (!Regex.Match(column, ColumnWithDataTypeRegex).Success)
```

```
                {
                    ret += column + "(string)";
                }
                else
                {
                    ret += column;
                }
            }
            return ret;
        }
    }
}
```

2. In the Solution Explorer, right-click the CustomDataProviderDemo project and select **Add Reference**. In the **Reference Manager** dialog that appears, on the Projects tab, select **CustomDataProvider** and click **OK**.
3. Run the project, and follow the instructions in the RichTextBox to see the custom data provider in action.

Section Report Walkthroughs

Section Report walkthroughs cover scenarios to introduce the key features of code-based and XML-based section reports. Learn about different section report walkthroughs categorized as follows.

Data
: This section contain the walkthroughs that explain various ways of working with data sources.

Layout
: This section contain the walkthroughs that explain how to create different section report layouts.

Chart
: This section contain the walkthroughs that demonstrate how to work with the ActiveReports Chart control in a section report.

Export
: This section contains the walkthrough that demonstrates how to set up report custom exporting to PDF, Excel, TIFF, RTF, and plain text formats.

Script
: This section contain the walkthroughs that demonstrate how to embed script in reports so that code becomes portable when you save a report layout to XML-based RPX format.

Parameters
: This section contain the walkthroughs that demonstrate how to use parameters in SubReport and Chart controls.

Web
: This section contains the walkthroughs that explain how to create a simple web service for each scenario and how to create a Document Windows application.

Data

This section contains the following walkthroughs that fall under the Data category.

Basic Data Bound Reports
: This walkthrough demonstrates the basics of setting up bound section reports.

Basic XML-Based Reports (RPX)
: This walkthrough demonstrates how to create a simple section report, using the XML-based report template.

Run-Time Data Sources
: Describes how to change the report data source at run time using the ReportStart event.

Bind a Section Report to CSV Data Source
: Describes how to bind a report to a CSV data source.

Basic Data Bound Reports

In ActiveReports, the simplest reporting style is a tabular listing of fields from a data source.

This walkthrough illustrates the basics of setting up bound reports by introducing the ideas of using the DataSource icon and dragging fields from the Report Explorer onto the report.

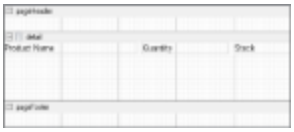
The walkthrough is split up into the following activities:

- Adding an ActiveReport to the Visual Studio project
- Connecting to a data source
- Adding controls to the report
- Viewing the report

> **Note:** This walkthrough uses the Northwind database. The NWIND.mdb file can be downloaded from GitHub: ..\Samples14\Data\NWIND.mdb.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout



Run-Time Layout

To add an ActiveReport to the Visual Studio project

1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 Section Report (code-based)** and in the Name field, rename the file as **rptBound**.
4. Click the **Add** button to open a new section report in the designer.

See Quick Start for information on adding different report layouts.

To connect the report to a data source

1. On the detail section band, click the Data Source Icon.

2. In the Report Data Source dialog that appears, on the **OLE DB** tab, next to Connection String, click the Build button.
3. In the Data Link Properties window that appears, select **Microsoft Jet 4.0 OLE DB Provider** and click the **Next** button to move to the Connection tab.
4. Click the ellipsis (...) button to browse to your database, for example the NWind.mdb sample database. Click **Open** once you have selected the appropriate database path.
5. Click the **Test Connection** button to see if you have successfully connected to the database.
6. Click **OK** to close the Data Link Properties window and return to the Report Data Source dialog. Notice that the Connection String field gets filled automatically.
7. In the **Query** field on the **OLE DB** tab, enter the following SQL query.

SQL Query
`SELECT * FROM Products`

8. Click **OK** to save the data source and return to the report design surface.

To create a layout for the report

1. In the Visual Studio toolbox, expand the **ActiveReports 14 Section Report** node and drag three TextBox controls onto the detail section and set the properties of each textbox as indicated:

TextBox1

Property Name	Property Value
DataField	ProductName
Text	Product Name
Location	0, 0in
Size	2.3, 0.2in

TextBox2

Property Name	Property Value
DataField	QuantityPerUnit
Text	Quantity
Location	2.4, 0in
Size	1.5, 0.2in

TextBox3

Property Name	Property Value
DataField	UnitsInStock
Text	Stock

Location	4, 0in
Size	1, 0.2in

2. Click just below the fields to select the Detail section, and in the Properties Window, set the **CanShrink** property to **True** to eliminate white space in the rendered report.

To view the report

- Click the preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

Basic XML-Based Reports (RPX)

ActiveReports allows you to create reports with embedded script and save them to the XML-based RPX file format. By embedding script in reports saved as RPX files, you can later load, run, and display reports directly in the viewer control without rebuilding the application. This walkthrough illustrates how to create a simple report, using the XML-based report template.

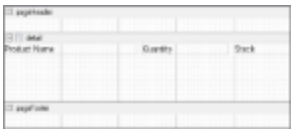
This walkthrough is split into the following activities:

- Adding an ActiveReport to the Visual Studio project
- Creating a layout for the report
- Adding scripting to supply data for the controls
- Applying scripting to set alternate row colors in the detail section
- Loading an XML-based report from resources

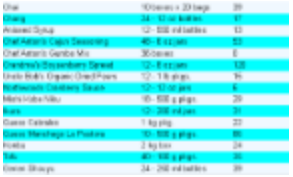
> **Note:** This walkthrough uses the Northwind database. The NWIND.mdb file can be downloaded from GitHub: ..\Samples14\Data\NWIND.mdb.

When you have finished this walkthrough, you get a report that looks similar to the following at design time and at run time.

Design-Time Layout



Run-Time Layout



To add an ActiveReport to the Visual Studio project

1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 Section Report (xml-based)** and in the Name field, rename the file as **rptScript**.
4. Click the **Add** button to open a new section report in the designer.
5. In the Solution Explorer click the **rptScript.rpx** item and in the properties window set the **Build Action** property to Embedded Resource.

See Quick Start for information on adding different report layouts.

To create a layout for the report

1. In the Visual Studio toolbox, expand the **ActiveReports 14 Section Report** node and drag three TextBox controls onto the detail section and set the properties of each textbox as indicated:

   TextBox1

Property Name	Property Value
DataField	ProductName
Text	Product Name
Location	0, 0in
Size	2.3, 0.2in

   TextBox2

Property Name	Property Value
DataField	QuantityPerUnit
Text	Quantity
Location	2.4, 0in
Size	2.4, 0.2in

   TextBox3

Property Name	Property Value
DataField	UnitsInStock
Text	Stock
Location	4, 0in
Size	1, 0.2in

2. Click just below the fields to select the Detail section, and in the Properties Window, set the **CanShrink** property to **True** to eliminate white space in the rendered report.

To add scripting to the report to supply data for the controls

1. Click the **Script** tab located at the bottom of the report designer to access the script editor.

2. Add the scripting code.

   The following example shows what the scripting code looks like.

> ⚠ **Warning:** Do not access the Fields collection outside the **DataInitialize** and **FetchData** events. Accessing the Fields collection outside of these events is not supported, and has unpredictable results.

To write the script in Visual Basic.NET

Visual Basic.NET script. Paste in the script editor window.

```
Private Shared m_cnn As System.Data.OleDb.OleDbConnection

Public Sub ActiveReport_ReportStart()
  'Set up a data connection for the report
  Dim connString As String = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=[User
Folder]\Samples14\Data\NWIND.mdb"
  Dim sqlString As String = "SELECT * FROM products"

  m_cnn = new System.Data.OleDb.OleDbConnection(connString)
  Dim m_Cmd As System.Data.OleDb.OleDbCommand = new
System.Data.OleDb.OleDbCommand(sqlString, m_cnn)

  If m_cnn.State = System.Data.ConnectionState.Closed Then
     m_cnn.Open
  End If
  rpt.DataSource = m_Cmd.ExecuteReader
End Sub

Public Sub ActiveReport_ReportEnd()
  'Close the data reader and connection
  m_cnn.Close
End Sub
```

To write the script in C#

C# script. Paste in the script editor window.

```
private static System.Data.OleDb.OleDbConnection m_cnn;

public void ActiveReport_ReportStart()
{
    //Set up a data connection for the report
    string m_cnnString = @"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=[User
Folder]\Samples14\Data\NWIND.mdb";
    string sqlString = "SELECT * FROM products";
    m_cnn = new System.Data.OleDb.OleDbConnection(m_cnnString);
    System.Data.OleDb.OleDbCommand m_Cmd = new
System.Data.OleDb.OleDbCommand(sqlString, m_cnn);
```

```
    if(m_cnn.State == System.Data.ConnectionState.Closed)
    {
        m_cnn.Open();
    }
    rpt.DataSource = m_Cmd.ExecuteReader();
}

public void ActiveReport_ReportEnd()
{
    //Close the data reader and connection
    m_cnn.Close();
}
```

To add scripting to alternate colors in the detail section

1. Click the **Script** tab located at the bottom edge of the report designer to access the scripting editor.
2. Add the scripting code to set alternate colors in the rows of the detail section.

   The following example shows what the scripting code looks like.

   To write the script in Visual Basic.NET

Visual Basic.NET script. Paste in the script editor window.

```
Dim b as boolean = true

Sub Detail_Format
 if b then
  Me.Detail.BackColor = Color.AliceBlue
  b= false
  else
  me.Detail.BackColor = Color.Cyan
  b = true
 End If
End Sub
```

   To write the script in C#

C# script. Paste in the script editor window.

```
bool color = true;
public void Detail_Format()
{
  if(color)
  {
      this.Detail.BackColor = System.Drawing.Color.AliceBlue;
      color = false;
  }
  else
```

```
  {
      this.Detail.BackColor = System.Drawing.Color.Cyan;
      color = true;
  }
}
```

Loading the report to the Viewer

You can quickly view your report at design time by clicking the Preview tab at the bottom of the designer. You can also load the report to the Viewer control.

- Drag the ActiveReports Viewer control from the Visual Studio toolbox onto the Windows Form and set its Dock property to Fill.
- Double-click the title bar of the Windows Form containing the viewer to create a Form_Load event and add the code needed to load the RPX into a generic ActiveReport and display it in the viewer.

  The following example shows what the code for the method looks like.

  To write the script in Visual Basic.NET

  Visual Basic.NET script. Paste INSIDE the Form_Load event.

```
Dim sectionReport As New GrapeCity.ActiveReports.SectionReport()
Dim xtr As New System.Xml.XmlTextReader("..\..\rptScript.rpx")
sectionReport.LoadLayout(xtr)
xtr.Close()
Viewer1.LoadDocument(sectionReport)
```

  To write the script in C#

  C# script. Paste INSIDE the Form_Load event.

```
GrapeCity.ActiveReports.SectionReport sectionReport = new
GrapeCity.ActiveReports.SectionReport();
System.Xml.XmlTextReader xtr = new
System.Xml.XmlTextReader(@"..\..\rptScript.rpx");
sectionReport.LoadLayout(xtr);
xtr.Close();
viewer1.LoadDocument(sectionReport);
```

Run-Time Data Sources

ActiveReports allows you to change the data source of a report at run time. This walkthrough illustrates how to change the data source at run time.

This walkthrough is split up into the following activities:

- Adding an ActiveReports to the Visual Studio project
- Connecting the report to a design time data source
- Adding controls to the report to display data
- Adding code to change the data source at run time

- Adding code to close the data connection
- Viewing the report

> **Note:** This walkthrough uses the Northwind database. The NWIND.mdb file can be downloaded from GitHub: ..\Samples14\Data\NWIND.mdb.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout

Run-Time Layout

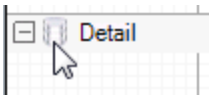
To add an ActiveReport to the Visual Studio project

1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 Section Report (code-based)** and in the Name field, rename the file as **rptModifyDS**.
4. Click the **Add** button to open a new section report in the designer.

See Quick Start for information on adding different report layouts.

To connect the report to a data source

> **Tip**: Even if you will change the data source at run time, setting a design time data source allows you to drag fields onto the report from the Report Explorer.

1. On the detail section band, click the Data Source Icon.
   

2. In the Report Data Source dialog that appears, on the **OLE DB** tab, next to Connection String, click the Build button.
3. In the Data Link Properties window that appears, select **Microsoft Jet 4.0 OLE DB Provider** and click the **Next** button to move to the Connection tab.
4. Click the ellipsis (...) button to browse to your database, for example the NWind.mdb sample database. Click **Open** once you have selected the appropriate database path.
5. Click the **Test Connection** button to see if you have successfully connected to the database.
6. Click **OK** to close the Data Link Properties window and return to the Report Data Source dialog. Notice that the Connection String field gets filled automatically.
7. In the **Query** field on the **OLE DB** tab, enter the following SQL query.

   ```
   SQL Query
   ```

```
SELECT * FROM Products
```

8. Click **OK** to save the data source and return to the report design surface.

To create a layout for the report

1. On the design surface of the report, select the detail section and in the Properties window, set the **CanShrink** property to **True**.
2. In the Report Explorer, expand the **Fields** node, then the **Bound** node. Drag the following fields onto the detail section and in the Properties window, set the following properties.

TextBox1 (ProductID)

Property Name	Property Value
Location	0, 0 in
Size	0.5, 0.2 in

TextBox2 (ProductName)

Property Name	Property Value
Location	0.6, 0 in
Size	2.8, 0.2 in

TextBox3 (UnitsInStock)

Property Name	Property Value
Location	3.5, 0 in
Size	0.5, 0.2 in
Alignment	Right

TextBox4 (UnitsOnOrder)

Property Name	Property Value
Location	4.1, 0 in
Size	0.5, 0.2 in
Alignment	Right

TextBox5 (UnitPrice)

Property Name	Property Value
Location	4.7, 0 in
Size	0.9, 0.2 in
Alignment	Right

OutputFormat	Currency

To change the data source at run time

To change the data source at run time

1. Double-click in the gray area below **rptModifyDS** to create an event-handling method for the **ReportStart** event.
2. Add code to the handler to change the data source at run time.

To write the code in Visual Basic.NET

The following example shows what the code for the method looks like.

Visual Basic.NET code. Paste JUST ABOVE the ReportStart event.

```
Dim conn As System.Data.OleDb.OleDbConnection
Dim reader As System.Data.OleDb.OleDbDataReader
```

Visual Basic.NET code. Paste INSIDE the ReportStart event.

```
Dim connString As String = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" + "[User
Folder]\Samples14\Data\NWIND.mdb"
conn = New System.Data.OleDb.OleDbConnection(connString)
Dim cmd As New System.Data.OleDb.OleDbCommand("SELECT * FROM Products WHERE UnitPrice =
18", conn)
conn.Open()
reader = cmd.ExecuteReader()
Me.DataSource = reader
```

To write the code in C#

The following example shows what the code for the method looks like.

C# code. Paste JUST ABOVE the ReportStart event.

```
private static System.Data.OleDb.OleDbConnection conn;
private static System.Data.OleDb.OleDbDataReader reader;
```

C# code. Paste INSIDE the ReportStart event.

```
string connString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" + @"[User
Folder]\Samples14\Data\NWIND.mdb";
conn = new System.Data.OleDb.OleDbConnection(connString);
System.Data.OleDb.OleDbCommand cmd = new System.Data.OleDb.OleDbCommand("SELECT * FROM
Products WHERE UnitPrice = 18", conn);
conn.Open();
reader = cmd.ExecuteReader();
this.DataSource = reader;
```

To close the data connection

To write the code in Visual Basic

1. In design view of rptModifyDS, drop down the field at the top left of the code view and select **(rptModifyDS Events)**.
2. Drop down the field at the top right of the code view and select **ReportEnd**. This creates an event-handling method for ReportEnd event.
3. Add code to the handler to close the data connection.

The following example shows what the code for the method looks like.

Visual Basic.NET code. Paste INSIDE the ReportEnd event.

```
reader.Close()
conn.Close()
```

To write the code in C#

1. Click in the gray area below rptModifyDS to select the report.
2. Click the events icon in the Properties Window to display available events for the report.
3. Double-click **ReportEnd**. This creates an event-handling method for the ReportEnd event.
4. Add code to the handler to close the data connection.

The following example shows what the code for the method looks like.

C# code. Paste INSIDE the ReportEnd event.

```
reader.Close();
conn.Close();
```

To view the report

- Click the preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

Bind a Section Report to CSV Data Source

This walkthrough illustrates binding a section report to a CSV data source.

The walkthrough is split up into the following activities:

- Adding an ActiveReport to the Visual Studio project
- Connecting to a data source
- Creating a layout for the report
- Viewing the report

> **Note:** This walkthrough uses the **Products_header_tab.csv** sample database. The Products_header_tab.csv file can be downloaded from GitHub: ..\Samples14\Data\NWIND.mdb.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout

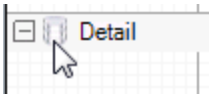
Run-Time Layout

To add an ActiveReport to the Visual Studio project

1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 Section Report (code-based)** and in the Name field, rename the file as **rptProductsStock**.
4. Click the **Add** button to open a new section report in the designer.

See Quick Start for information on adding different report layouts.

To connect the report to a data source

1. On the detail section band, click the Data Source icon.


2. In the Report Data Source dialog, on the **CSV** tab, click the **Build** button next to Connection String.
3. To specify the **File Path**, click the **Open** button and navigate to the Products_header_tab.csv file.
4. Select the **Column Separator** as **Tab** from the drop-down menu. See the **Sample CSV Connection String** drop-down in CSV Data Provider topic for further details.
5. Click **OK** to save the changes and close the **Configure CSV Data Source** wizard. The **Connection String** tab displays the generated connection string as shown below:
```
Path=C:\\[User folder]\\Samples14\\Data\\Products_header_tab.csv;Locale=en-US;
TextQualifier=";ColumnsSeparator= ;RowsSeparator=\r\n;HasHeaders=True
```
6. Click **OK** to close the Report Data Source dialog. You have successfully connected the report to the CSV data source.

To view the report

- Click the preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

Layout

This section contains the following walkthroughs that fall under the Layout category.

Address Labels
: This walkthrough demonstrates how to create a report that repeats labels using the LayoutAction property.

Columnar Reports
: This walkthrough demonstrates how to create a simple report using columns.

Group On Unbound Fields
: This walkthrough demonstrates how to set up grouping in an unbound section report.

Mail Merge with RichText
: This walkthrough demonstrates how to create a mail-merge report using the RichText control.

Overlaying Reports (Letterhead)
: This walkthrough demonstrates how to overlay an ActiveReport with a static letterhead report.

Run-Time Layouts
: Describes how to create and modify report layouts dynamically.

Subreports with XML Data
: Learn how to use XML data with subreports.

Subreports with Run-Time Data Sources
: Learn how to embed a subreport in a main report, passing the data source from the main report to the subreport at run time.

Address Labels

ActiveReports can be used to print any label size by using the newspaper column layout.

This walkthrough illustrates how to create a report that repeats labels using the LayoutAction property and prints labels to a laser printer. The labels in this example are 1" x 2.5" and print 30 labels per 8½" x 11" sheet.

The walkthrough is split up into the following activities:

- Connecting the report to a data source
- Adding controls to the report to display data
- Adding code to the detail_Format event to repeat labels
- Viewing the report

> **Note:** This walkthrough uses the Northwind database. The NWIND.mdb file can be downloaded from GitHub: ..\Samples14\Data\NWIND.mdb.

When you have finished this walkthrough, you get a report that looks similar to the following at design time and at run time.

Design-Time Layout

Run-Time Layout

To add an ActiveReport to the Visual Studio project

1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 Section Report (code-based)** and in the Name field, rename the file as **rptLabels**.
4. Click the **Add** button to open a new section report in the designer.

See Quick Start for information on adding different report layouts.

To connect the report to a data source

1. On the detail section band, click the Data Source Icon.


2. In the Report Data Source dialog that appears, on the **OLE DB** tab, next to Connection String, click the Build button.
3. In the Data Link Properties window that appears, select **Microsoft Jet 4.0 OLE DB Provider** and click the **Next** button to move to the Connection tab.
4. Click the ellipsis (...) button to browse to your database, for example the NWind.mdb sample database. Click **Open** once you have selected the appropriate database path.
5. Click the **Test Connection** button to see if you have successfully connected to the database.
6. Click **OK** to close the Data Link Properties window and return to the Report Data Source dialog. Notice that the Connection String field gets filled automatically.
7. In the **Query** field on the **OLE DB** tab, enter the following SQL query.

SQL Query
`SELECT ContactName, CompanyName, Address, City, PostalCode, Country` `FROM Customers`

8. Click **OK** to save the data source and return to the report design surface.

To create a layout for the report

1. Right-click the PageHeader section and select **Delete** to remove the PageHeader and Footer sections from the report.
2. In the Report menu, select **Settings** and change the margins as follows:
    - Top margin: 0.5
    - Bottom margin: 0.5
    - Left margin: 0.2
    - Right margin: 0.2
3. In the Report Explorer, select **Report** and in the Properties Window, set the **PrintWidth** property to **8.1** (the width of the label sheet less the Left and Right margins).
4. Click the detail section of the report to select it and in the Properties window, set the properties as follows.

Property Name	Property Value

CanGrow	False
ColumnCount	3
ColumnDirection	AcrossDown
ColumnSpacing	0.2
Height	1

5. From the toolbox, drag six TextBox controls onto the detail section and set the properties of each textbox as follows.

TextBox1

Property Name	Property Value
DataField	ContactName
Location	0, 0 in
Size	2.5, 0.2 in
Font > Bold	True

TextBox2

Property Name	Property Value
DataField	CompanyName
Location	0, 0.2 in
Size	2.5, 0.2 in

TextBox3

Property Name	Property Value
DataField	Address
Location	0, 0.4 in
Size	2.5, 0.2 in

TextBox4

Property Name	Property Value
DataField	City
Location	0, 0.6 in
Size	2.5, 0.2 in

TextBox5

Property Name	Property Value

DataField	PostalCode
Location	0, 0.8 in
Size	1.45, 0.2 in

TextBox6

Property Name	Property Value
DataField	Country
Location	1.5, 0.8 in
Size	1, 0.2 in

6. Select all of the textboxes, and in the Properties Window, set the **CanGrow** property to **False**. This prevents overlapping text, but may crop data if one of the fields contains more data than the control size allows.

If you preview the report at this point, one copy of each label appears on the page.

To add code to the detail_Format event to repeat labels

1. Double-click in the detail section to create a detail_Format event.
2. Add the following code to the event to repeat each label across all three columns.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the Format event.

```
'print each label three times
Static counter As Integer
counter = counter + 1
If counter <= 2 Then
    Me.LayoutAction = GrapeCity.ActiveReports.LayoutAction.MoveLayout Or
 GrapeCity.ActiveReports.LayoutAction.PrintSection
Else
    Me.LayoutAction = GrapeCity.ActiveReports.LayoutAction.MoveLayout Or
 GrapeCity.ActiveReports.LayoutAction.NextRecord Or
 GrapeCity.ActiveReports.LayoutAction.PrintSection
    counter = 0
End If
```

To write the code in C#

C# code. Paste JUST ABOVE the Format event.

```
int counter=0;
```

C# code. Paste INSIDE the Format event.

```
//print each label three times
counter = counter + 1;
if (counter <= 2)
```

```
{
    this.LayoutAction =  GrapeCity.ActiveReports.LayoutAction.MoveLayout|
GrapeCity.ActiveReports.LayoutAction.PrintSection;
}
else
{
    this.LayoutAction = GrapeCity.ActiveReports.LayoutAction.MoveLayout|
GrapeCity.ActiveReports.LayoutAction.NextRecord|
GrapeCity.ActiveReports.LayoutAction.PrintSection;
    counter = 0;
}
```

To view the report

- Click the preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

Columnar Reports

ActiveReports supports newspaper column layouts in both the Detail and Group sections. You can render the columns either horizontally or vertically in the section with options to break the column on the Group section (i.e. start a new column on the change of a group).

There is also a Boolean **ColumnGroupKeepTogether** property on the **GroupHeader**. When set to True, the ColumnGroupKeepTogether property attempts to prevent a group from splitting across columns. If a group cannot fit in the current column, it tries the next. If the group is too large for a single column, the property is ignored.

> **Note**: The **ColumnGroupKeepTogether** property only works when the GroupHeader's **GroupKeepTogether** property is set to **All**.

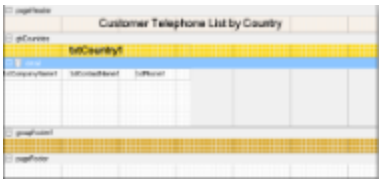
This walkthrough illustrates how to create a simple report using columns, and is split up into the following activities:

- Connecting the report to a data source
- Adding controls to the report to display data
- Viewing the report

> **Note:** This walkthrough uses the Northwind database. The NWIND.mdb file can be downloaded from GitHub: ..\Samples14\Data\NWIND.mdb.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout



Run-Time Layout

To add an ActiveReport to the Visual Studio project

1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 Section Report (code-based)** and in the Name field, rename the file as **rptColumnar**.
4. Click the **Add** button to open a new section report in the designer.

See Quick Start for information on adding different report layouts.

To connect the report to a data source

1. On the detail section band, click the Data Source Icon.
   

2. In the Report Data Source dialog that appears, on the **OLE DB** tab, next to Connection String, click the Build button.
3. In the Data Link Properties window that appears, select **Microsoft Jet 4.0 OLE DB Provider** and click the **Next** button to move to the Connection tab.
4. Click the ellipsis (...) button to browse to your database, for example the NWind.mdb sample database. Click **Open** once you have selected the appropriate database path.
5. Click the **Test Connection** button to see if you have successfully connected to the database.
6. Click **OK** to close the Data Link Properties window and return to the Report Data Source dialog. Notice that the Connection String field gets filled automatically.
7. In the **Query** field on the **OLE DB** tab, enter the following SQL query.

SQL Query
`SELECT Country, CompanyName, ContactName, Phone FROM Customers ORDER BY Country`

8. Click **OK** to save the data source and return to the report design surface.

To create a layout for the report

1. Right-click the design surface of the report and select **Insert**, then **Group Header/Footer** to add a GroupHeader/Footer section.
2. Select the group header and in the Properties Window, set the properties as follows.

Property Name	Property Value

Name	ghCountry
BackColor	Gold
DataField	Country
ColumnGroupKeepTogether	True
GroupKeepTogether	All

3. Select the group footer and in the Properties window, set the **BackColor** property to **Goldenrod**.
4. In the Report Explorer, drag the **Country** field onto the GroupHeader section and in the Properties window, set its properties as follows.

Property Name	Property Value
Location	0, 0 in
Size	3.25, 0.2 in
Alignment	Center
Font > Size	12
Font > Bold	True

5. Select the PageHeader section and in the Properties window, set the **BackColor** property to **Linen**.
6. From the toolbox, drag a Label control onto the PageHeader section and in the Properties window, set the properties as follows:

Property Name	Property Value
Location	0, 0 in
Size	6.5, 0.25 in
Alignment	Center
Font > Size	14
Text	Customer Telephone List by Country

7. Select the Detail section and in the Properties window, set the properties as follows.

Property Name	Property Value
CanShrink	True
ColumnCount	2

8. In the Report Explorer, expand the **Fields** node, then the **Bound** node. Drag the following fields onto the Detail section and set the properties of each textbox as indicated.

TextBox1

Property Name	Property Value
DataField	CompanyName
Location	0, 0 in
Size	1.15, 0.2 in
Font > Size	8pt

TextBox2

Property Name	Property Value
DataField	ContactName
Location	1.15, 0 in
Size	1.15, 0.2 in
Font > Size	8pt

TextBox3

Property Name	Property Value
DataField	Phone
Location	2.3, 0 in
Size	0.95, 0.2 in
Font > Size	8pt

To view the report

- Click the preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

Group On Unbound Fields

ActiveReports allows you to set up grouping in unbound reports. When setting up grouping, the group header's DataField property is used to retrieve the grouping data from the database in the same manner as a textbox's DataField property. This walkthrough illustrates how to set up grouping in an unbound report.

This walkthrough is split into the following activities:

- Adding code to connect the report to a data source
- Adding controls to contain the data
- Using the DataInitialize event to add fields to the report's fields collection
- Using the FetchData event to populate the report fields
- Adding code to close the connection to the data source
- Viewing the report

> **Note:** This walkthrough uses the Northwind database. The NWIND.mdb file can be downloaded from GitHub: ..\Samples14\Data\NWIND.mdb.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout

Run-Time Layout

To add an ActiveReport to the Visual Studio project

1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 Section Report (code-based)** and in the Name field, rename the file as **rptGroupUnbound**.
4. Click the **Add** button to open a new section report in the designer.

See Quick Start for information on adding different report layouts.

To add code to connect the report to a data source

1. Double-click the gray area below the report. This creates an event-handling method for the report's **ReportStart** event.
2. Add code to the handler to:
    - Set the data source connection string
    - Set the data source SQL query
    - Open the connection and retrieve the data with the data reader

The following examples show what the code for the method looks like in Visual Basic.NET and C#.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste JUST ABOVE the ReportStart event.

```
Dim connection As System.Data.OleDb.OleDbConnection
Dim reader As System.Data.OleDb.OleDbDataReader
```

Visual Basic.NET code. Paste INSIDE the ReportStart event.

```
'Create the data connection and change the data source path as necessary
Dim connectionString As String
connectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=[User
Folder]\Samples14\Data\NWIND.mdb"
```

```
connection = New System.Data.OleDb.OleDbConnection(connectionString)
connection.Open()

Dim sqlString As String
sqlString = "SELECT * FROM categories INNER JOIN products ON categories.categoryid=
products.categoryid ORDER BY categories.CategoryID"
Dim command As New System.Data.OleDb.OleDbCommand(sqlString, connection)
'Retrieve data
reader = command.ExecuteReader()
```

To write the code in C#

C# code. Paste JUST ABOVE the ReportStart event.

```
private System.Data.OleDb.OleDbConnection connection;
private System.Data.OleDb.OleDbDataReader reader;
```

C# code. Paste INSIDE the ReportStart event.

```
//Create the data connection and change the data source path as necessary
string connectionString = @"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=[User
Folder]\Samples14\Data\NWIND.mdb";
connection=new System.Data.OleDb.OleDbConnection(connectionString);
connection.Open();

string sqlString = "SELECT * FROM categories INNER JOIN products ON
categories.categoryid = products.categoryid ORDER BY categories.CategoryID";
System.Data.OleDb.OleDbCommand command = new System.Data.OleDb.OleDbCommand(sqlString,
connection);

//Retrieve data
reader = command.ExecuteReader();
```

To create a layout for the report

1. On the design surface of the report, right-click and select **Insert**, then **Group Header/Footer** to add group header and footer sections.
2. Select the group header and in the Properties window, set the properties as follows.

Property Name	Property Value
Name	ghCategories
BackColor	Silver
CanShrink	True
DataField	CategoryID
GroupKeepTogether	All
KeepTogether	True

3. Select the group footer, and in the Properties Window, change the **Name** property to **gfCategories**.

4. Select the Detail section, and in the Properties Window, change the **CanShrink** property to **True**.
5. From the toolbox, drag the following controls to the Group Header section (drag the bottom edge of the section down to display all of the controls) and in the Properties window, set the properties of each control as follows.

TextBox1

Property Name	Property Value
DataField	CategoryName
Name	txtCategoryName
Text	Category Name
Location	0, 0 in
Size	2, 0.2 in
ForeColor	Blue
BackColor	Silver
Font > Size	12
Font > Bold	True

TextBox2

Property Name	Property Value
DataField	Description
Name	txtDescription
Text	Description
Location	0, 0.3 in
Size	6, 0.2 in

Label1

Property Name	Property Value
Name	lblProductName
Text	Product Name
Location	0, 0.6 in
Font > Bold	True

Label2

Property Name	Property Value
Name	lblUnitsInStock
Text	Units In Stock

Location	4.4, 0.6 in
Font > Bold	True
Alignment	Right

6. From the toolbox, drag two Textbox controls to the Detail section and in the Properties window, set the properties of each control as follows.

TextBox1

Property Name	Property Value
DataField	ProductName
Name	txtProductName
Text	Product Name
Location	0, 0 in
Size	4, 0.2 in

TextBox2

Property Name	Property Value
DataField	UnitsInStock
Name	txtUnitsInStock
Text	Units In Stock
Location	4.4, 0 in
Alignment	Right

7. From the toolbox, drag the following controls to the Group Footer section and in the Properties window, set the properties of each control as follows.

Label

Property Name	Property Value
DataField	TotalLabel
Name	lblTotalLabel
Location	2, 0 in
Size	2.4, 0.2 in

TextBox

Property Name	Property Value
DataField	ProductName

Name	txtTotalItems
Text	Total Items
Location	4.4, 0 in
SummaryType	SubTotal
SummaryFunc	Count
SummaryRunning	Group
SummaryGroup	ghCategories
Alignment	Right

Line

Property Name	Property Value
Name	Line1
LineWeight	3
X1	1.2
X2	6.45
Y1	0
Y2	0

8. Right-click the Page Header section and select **Delete**.

To add fields using the DataInitialize event

> ⚠ **Warning**: Do not access the Fields collection outside the **DataInitialize** and **FetchData** events. Accessing the Fields collection outside of these events is not supported, and has unpredictable results.

To write the code in Visual Basic

1. Right-click in any section of the design surface of the report, and select **View Code** to display the code view for the report.
2. At the top left of the code view of the report, click the drop-down arrow and select **(YourReportName Events)**.
3. At the top right of the code window, click the drop-down arrow and select **DataInitialize**. This creates an event-handling method for the report's DataInitialize event.
4. Add code to the handler to add fields to the report's Fields collection.

The following example shows what the code for the method looks like.

Visual Basic.NET code. Paste INSIDE the DataInitialize event.

```
Fields.Add("CategoryID")
Fields.Add("CategoryName")
Fields.Add("ProductName")
Fields.Add("UnitsInStock")
```

```
Fields.Add("Description")
Fields.Add("TotalLabel")
```

To write the code in C#

1. Click in the gray area below the report to select it.
2. Click the events icon in the Properties Window to display available events for the report.
3. Double-click **DataInitialize**. This creates an event-handling method for the report's DataInitialize event.
4. Add code to the handler to add fields to the report's Fields collection.

The following example shows what the code for the method looks like.

C# code. Paste INSIDE the DataInitialize event.

```
Fields.Add("CategoryID");
Fields.Add("CategoryName");
Fields.Add("ProductName");
Fields.Add("UnitsInStock");
Fields.Add("Description");
Fields.Add("TotalLabel");
```

To populate the fields using the FetchData event

To write the code in Visual Basic

1. At the top left of the code view for the report, click the drop-down arrow and select **(*YourReportName* Events)**.
2. At the top right of the code window, click the drop-down arrow and select **FetchData**. This creates an event-handling method for the report's FetchData event.
3. Add code to the handler to retrieve information to populate the report fields.

The following example shows what the code for the method looks like.

Visual Basic.NET code. Paste INSIDE the FetchData event.

```
Try
   reader.Read()
   Me.Fields("CategoryID").Value = reader("categories.CategoryID")
   Me.Fields("CategoryName").Value = reader("CategoryName")
   Me.Fields("ProductName").Value = reader("ProductName")
   Me.Fields("UnitsInStock").Value = reader("UnitsInStock")
   Me.Fields("Description").Value = reader("Description")
   Me.Fields("TotalLabel").Value = "Total Number of " + reader("CategoryName") + ":"
   eArgs.EOF = False
Catch
   eArgs.EOF = True
End Try
```

To write the code in C#

1. Back in design view, click in the gray area below the report to select it.
2. Click the events icon in the Properties window to display available events for the report.

3. Double-click **FetchData**. This creates an event-handling method for the report's FetchData event.
4. Add code to the handler to retrieve information to populate the report fields.

The following example shows what the code for the method looks like.

C# code. Paste INSIDE the FetchData event.

```
try
   {
      reader.Read();
      Fields["CategoryID"].Value = reader["categories.CategoryID"].ToString();
      Fields["CategoryName"].Value = reader["CategoryName"].ToString();
      Fields["ProductName"].Value = reader["ProductName"].ToString();
      Fields["UnitsInStock"].Value = reader["UnitsInStock"].ToString();
      Fields["Description"].Value = reader["Description"].ToString();
      Fields["TotalLabel"].Value = "Total Number of " +
reader["CategoryName"].ToString() + ":";
      eArgs.EOF = false;
   }
catch
   {
      eArgs.EOF = true;
   }
```

To add code to close the connection to the data source

To write the code in Visual Basic

1. At the top left of the code view for the report, click the drop-down arrow and select **(*YourReportName* Events)**.
2. At the top right of the code window, click the drop-down arrow and select **ReportEnd**. This creates an event-handling method for the report's ReportEnd event.
3. Add code to the handler to close the connection.

Visual Basic.NET code. Paste INSIDE the ReportEnd event.

```
reader.Close()
connection.Close()
```

To write the code in C#

1. Back in design view, click in the gray area below the report to select it.
2. Click the events icon in the Properties window to display available events for the report.
3. Double-click **ReportEnd**. This creates an event-handling method for the report's ReportEnd event.
4. Add code to the handler to close the connection.

The following example shows what the code for the method looks like.

C# code. Paste INSIDE the ReportEnd event.

```
reader.Close();
connection.Close();
```

To view the report

- Click the preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

Mail Merge with RichText

ActiveReports supports field merged reports using the RichText control. The RichText control can contain field place holders that can be replaced with values (merged) at run time. This walkthrough illustrates how to create a mail-merge report using the RichText control.

This walkthrough is split up into the following activities:

- Adding an ActiveReports to the Visual Studio project
- Connecting the report to a data source
- Adding controls and formatting the report
- Adding fields and text to the RichText control
- Using the FetchData event to conditionally format data
- Adding code to update RichText fields with current date and conditional values
- Adding code to send the group subtotal value to the RichText field
- Viewing the report

> **Note:** This walkthrough uses the Northwind database. The NWIND.mdb file can be downloaded from GitHub: ..\Samples14\Data\NWIND.mdb.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout

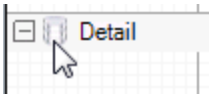
Run-Time Layout

To add an ActiveReport to the Visual Studio project

1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 Section Report (code-based)** and in the Name field, rename the file as **rptLetter**.
4. Click the **Add** button to open a new section report in the designer.

See Quick Start for information on adding different report layouts.

To connect the report to a data source

1. On the detail section band, click the Data Source Icon.


2. In the Report Data Source dialog that appears, on the **OLE DB** tab, next to Connection String, click the Build button.
3. In the Data Link Properties window that appears, select **Microsoft Jet 4.0 OLE DB Provider** and click the **Next** button to move to the Connection tab.
4. Click the ellipsis (...) button to browse to your database, for example the NWind.mdb sample database. Click **Open** once you have selected the appropriate database path.
5. Click the **Test Connection** button to see if you have successfully connected to the database.
6. Click **OK** to close the Data Link Properties window and return to the Report Data Source dialog. Notice that the Connection String field gets filled automatically.
7. In the **Query** field on the **OLE DB** tab, enter the following SQL query.

SQL Query
`SELECT Customers.CustomerID, Customers.CompanyName, Customers.ContactName, Customers.Address, Customers.City, Customers.Region, Customers.Country, Customers.PostalCode, Orders.OrderID, Orders.OrderDate, [Order Subtotals].Subtotal FROM Customers INNER JOIN ([Order Subtotals] INNER JOIN Orders ON [Order Subtotals].OrderID = Orders.OrderID) ON Customers.CustomerID = Orders.CustomerID`

8. Click **OK** to save the data source and return to the report design surface.

To create a layout for the report

1. On the design surface of the report, right-click and select **Insert**, then **Group Header/Footer** to add group header and footer sections.
2. On the design surface, select the grey area outside the report and in the Properties window, set the **PrintWidth** property to **6.5**.
3. Select the group header and in the Properties window, set the properties as follows.

Property Name	Property Value
DataField	CustomerID
Height	2.5
KeepTogether	True

4. On the design surface of the report, select the group footer section and in the Properties window, set the following properties.

Property Name	Property Value
Height	1.1
KeepTogether	True
NewPage	After

5. On the design surface of the report, select the detail section and in the Properties window, set the **CanShrink** property to **True**.
6. On the design surface of the report, select the pageHeader section and in the Properties window, set the following properties.

Property Name	Property Value
Height	0.8
BackColor	Coral

7. From the toolbox, drag the Label control to the pageHeader section and in the Properties window, set the properties as follows.

   Label

Property Name	Property Value
Location	0, 0 in
Size	6.5, 0.65 in
Text	GrapeCity
Font > Size	36
Font > Bold	True

8. In the Report Explorer, expand the **Fields** node, then the **Bound** node. Drag the **SubTotal** field onto the groupHeader section and in the Properties window, set the following properties.

Property Name	Property Value
Location	4, 0 in
Size	1, 0.2 in
Name	txtSubtotal1
OutputFormat	Currency
Visible	False
SummaryType	SubTotal

SummaryGroup	groupHeader1

> **Note**: Even though txtSubtotal1 is hidden, setting its properties is important as it provides the value and the formatting that is displayed in the RichText control.

9. From the toolbox, drag the following controls to the groupHeader section and in the Properties window, set the properties as follows.

RichTextBox

Property Name	Property Value
Location	0, 0 in
Size	6.5, 2.1 in
AutoReplaceFields	True

Label1

Property Name	Property Value
Location	0.875, 2.25 in
Size	1, 0.2 in
Text	Order ID
Font > Bold	True

Label2

Property Name	Property Value
Location	1.875, 2.25 in
Size	1, 0.2 in
Text	Order Date
Font > Bold	True

Label3

Property Name	Property Value
Location	4.375, 2.25 in
Size	1, 0.2 in
Text	Amount
Font > Bold	True
Alignment	Right

10. In the Report Explorer, expand the **Fields** node, then the **Bound** node. Drag the following fields onto the detail section and in the Properties window, set the properties of each textbox as follows.

TextBox1 (OrderID)

Property Name	Property Value
Location	0.875, 0 in
Size	1, 0.2 in

TextBox2 (OrderDate)

Property Name	Property Value
Location	1.875, 0 in
Size	1, 0.2 in
OutputFormat	Date (MM/dd/yy)

TextBox3 (Subtotal)

Property Name	Property Value
Location	4.375, 0 in
Size	1, 0.2 in
OutputFormat	Currency
Alignment	Right

11. From the toolbox, drag the following controls to the groupFooter section and in the Properties window, set the properties as follows.

Label1

Property Name	Property Value
Location	5.15, 0.15 in
Size	1.35, 0.2 in
Text	Best regards,
Alignment	Right

Label2

Property Name	Property Value
Location	5.15, 0.8 in
Size	1.35, 0.2 in
Text	Accounts Receivable

To add fields to the RichText control

1. Double-click the RichTextBox control box and delete the default text.
2. Right-click the box and choose **Insert Fields**.
3. In the **Insert Field** dialog that appears, enter **Date** and click **OK**.
4. Place the cursor in front of the text **[!Date]** that appears in the RichText control, and add spaces until the text is at the right edge of the control (but not overlapping to the next line).
5. Place the cursor at the end of the text, and press the **Enter** key to move to the next line.
6. Insert each of the following fields using the **Insert Field** dialog (see design time image above for fields arrangement):
    - CompanyName
    - ContactName
    - Address
    - City
    - Region
    - Country
    - PostalCode
    - SubTotal
7. Add the following text to the RichText control box after all of the fields.

Paste into the RichText control

```
Dear [!ContactName],

Thank you for your business. Below is a list of your orders for the past year with
a total of [!SubTotal].
Please take this opportunity to review each order and total for accuracy. Call us
at 1-800-DNT-CALL with
any questions or concerns.
```

8. Arrange the text and fields within the control as you would in any text editor.

To use the FetchData event to conditionally format data

To write the code in Visual Basic

1. At the top left of the code view for the report, click the drop-down arrow and select **(rptLetter Events)**.
2. At the top right of the code window, click the drop-down arrow and select **FetchData**. This creates an event-handling method for the report's FetchData event.
3. Add code to the handler to add a comma and a space if there is a Region value for the customer's address.

The following example shows what the code for the method looks like.

Visual Basic.NET code. Paste JUST ABOVE the FetchData event.

```
Dim region As String
```

Visual Basic.NET code. Paste INSIDE the FetchData event.

```
'If there is no region for the customer, display nothing
If Fields("Region").Value Is System.DBNull.Value Then
    region = ""
Else
'If there is a region, add a comma and a space
    region = ", " + Fields("Region").Value
End If
```

To write the code in C#

1. Back in design view, click in the gray area below the report to select it.
2. Click the events icon in the Properties window to display available events for the report.
3. Double-click **FetchData**. This creates an event-handling method for the report's FetchData event.
4. Add code to the handler to add a comma and a space if there is a Region value for the customer's address.

The following example shows what the code for the method looks like.

C# code. Paste JUST ABOVE the FetchData event.

```
string region;
```

C# code. Paste INSIDE the FetchData event.

```
if(Fields["Region"].Value is System.DBNull)
  region = "";
else
  region = ", " + Fields["Region"].Value.ToString();
```

To add code to update RichText fields with the current date and conditional values

1. Double-click in the group header section of the report to create an event-handling method for the group header's Format event.
2. Add code to the handler to:
    - Replace the Date field in the RichText control with the current system date
    - Replace the Region field with the conditional value created in the FetchData event

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the Group Header Format event.

```
'Use the current date in the letter
Me.RichTextBox1.ReplaceField("Date", System.DateTime.Today.Date.ToShortDateString())
'Use the value returned by the FetchData event
Me.RichTextBox1.ReplaceField("Region", region)
```

To write the code in C#

C# code. Paste INSIDE the Group Header Format event.

```
//Use the current date in the letter
this.richTextBox1.ReplaceField("Date", System.DateTime.Today.Date.ToShortDateString());
//Use the value returned by the FetchData event
this.richTextBox1.ReplaceField("Region", region);
```

To add code to send the group subtotal value to the RichText field

To write the code in Visual Basic.NET

1. Right-click in any section of the design window of rptLetter, and click on **View Code** to display the code view for the report.

2. At the top left of the code view for rptLetter, click the drop-down arrow and select **GroupHeader1**.
3. At the top right of the code window, click the drop-down arrow and select **BeforePrint**. This creates an event-handling method for rptLetter's GroupHeader1_BeforePrint event.

   > **Note:** We use the BeforePrint event instead of the Format event to get the final value of the subtotal field just prior to printing. For more information on section event usage, see the Section Events topic.

4. Add code to the handler to replace the value of the Subtotal field in the RichText control with the value of the hidden textbox in the group header.

   Visual Basic.NET code. Paste INSIDE the Group Header BeforePrint event.

   ```
   'Use the value from the hidden group subtotal field
   Me.RichTextBox1.ReplaceField("SubTotal", Me.txtSubtotal1.Text)
   ```

To write the code in C#

1. Back in design view, click the group header section to select it.
2. Click the events icon in the Properties window to display available events for the group header.
3. Double-click BeforePrint. This creates an event-handling method for the report's BeforePrint event.
   Add code to the handler to replace the value of the Subtotal field in the RichText control with the value of the hidden textbox in the group header.
   The following example shows what the code for the method looks like.

   C# code. Paste INSIDE the Group Header BeforePrint event.

   ```
   //Use the value from the hidden group subtotal field
   this.richTextBox1.ReplaceField("SubTotal", this.txtSubtotal1.Text);
   ```

To view the report

- Click the preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

Overlaying Reports (Letterhead)

ActiveReports allows you to overlay static report formats over data reports. This walkthrough illustrates how to overlay an ActiveReport with a static letterhead report.

This walkthrough is split up into the following activities:

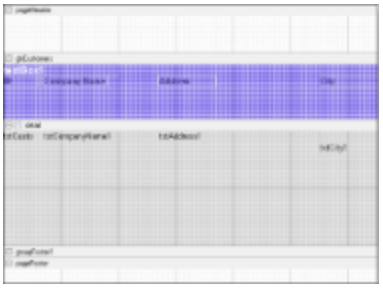
- Adding an ActiveReports to the Visual Studio project
- Connecting the data report to a data source
- Adding controls to the letterhead and data reports
- Adding code to overlay the data report pages with the letterhead report
- Viewing the report

> **Note:** This walkthrough uses the Northwind database. The NWIND.mdb file can be downloaded from GitHub: ..\Samples14\Data\NWIND.mdb.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout (rptLetterhead)

Design-Time Layout (rptData)



Run-Time Layout

To add an ActiveReport to the Visual Studio project

1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select ActiveReports 14 **Section Report (code-based)** and in the Name field, rename the file as **rptLetterhead**.
4. Click the **Add** button to open a new section report in the designer.
5. From the **Project** menu, select **Add New Item**.
6. In the Add New Item dialog that appears, select ActiveReports 14 **Section Report (code-based)** and in the Name field, rename the file as **rptData**.
7. Click the **Add** button to open a new section report in the designer.

See Quick Start for information on adding different report layouts.

To connect the rptData to a data source

1. On the detail section band, click the Data Source Icon.


2. In the Report Data Source dialog that appears, on the **OLE DB** tab, next to Connection String, click the Build button.
3. In the Data Link Properties window that appears, select **Microsoft Jet 4.0 OLE DB Provider** and click the **Next**

button to move to the Connection tab.
4. Click the ellipsis (...) button to browse to your database, for example the NWind.mdb sample database. Click **Open** once you have selected the appropriate database path.
5. Click the **Test Connection** button to see if you have successfully connected to the database.
6. Click **OK** to close the Data Link Properties window and return to the Report Data Source dialog. Notice that the Connection String field gets filled automatically.
7. In the **Query** field on the **OLE DB** tab, enter the following SQL query.

SQL Query
`SELECT * FROM Customers ORDER BY Country`

8. Click **OK** to save the data source and return to the report design surface.

To create a layout for the rptData

1. Select the **PageHeader** section and in the Properties Window, set the **Height** property to **0.65**. (This will match the height of the page header in the template.)
2. On the design surface, select the grey area outside the report and in the Properties window, set the **PrintWidth** property to **6.5**.
3. Right-click the report and select **Insert > GroupHeader/Footer** to add group header and group footer sections.
4. Select the group header and in the Properties window, set the properties as follows.

Property Name	Property Value
Name	ghCustomers
BackColor	MediumSlateBlue
CanShrink	True
DataField	Country
GroupKeepTogether	FirstDetail
KeepTogether	True

5. From the toolbox, drag the following controls to **ghCustomers** and in the Properties window, set the properties as follows.

TextBox1

Property Name	Property Value
DataField	="Customers in " + Country (DataField)
Size	2, 0.2 in
Location	0, 0 in
Font Bold	True
ForeColor	White
Font Size	12

Label1

Property Name	Property Value
Text	ID
Size	0.6, 0.2 in
Location	0, 0.2 in
Font Bold	True
ForeColor	DarkSlateBlue

Label2

Property Name	Property Value
Text	Company Name
Size	1.1, 0.2 in
Location	0.7, 0.2 in
Font Bold	True
ForeColor	DarkSlateBlue

Label3

Property Name	Property Value
Text	Address
Size	1, 0.2 in
Location	2.7, 0.2 in
Font Bold	True
ForeColor	DarkSlateBlue

Label4

Property Name	Property Value
Text	City
Size	1, 0.2 in
Location	5.5, 0.2 in
Font Bold	True
ForeColor	DarkSlateBlue

6. Click the **Detail** section and in the Properties window, set the properties as follows.

Property Name	Property Value
BackColor	LightGray
CanShrink	True

7. From the toolbox, drag four TextBox controls onto the **Detail** section and set the properties of each textbox as follows.

TextBox1

Property Name	Property Value
DataField	CustomerID
Size	0.6, 0.2 in
Location	0, 0 in

TextBox2

Property Name	Property Value
DataField	CompanyName
Size	2, 0.2 in
Location	0.7, 0 in

TextBox3

Property Name	Property Value
DataField	Address
Size	2.8, 0.2 in
Location	2.7, 0 in

TextBox4

Property Name	Property Value
DataField	City
Size	1, 0.2 in
Location	5.5, 0.2 in

8. Select the group footer and in the Properties window, set the **Height** property to **0**.

To create a layout for the rptLetterhead

1. Select the **Page Header** and in the Properties window, set the properties as follows.

Property Name	Property Value
BackColor	DarkSlateBlue
Height	0.65

2. From the toolbox, drag a Label control onto the **Page Header** and in the Properties window, set the properties as follows.

Label1

Property Name	Property Value
Size	6.5, 0.65 in
Location	0, 0 in
Font Size	36
Font Bold	True
ForeColor	White
Text	GrapeCity

3. Select the **Page Footer** and in the Properties window, set the **BackColor** property to **DarkSlateBlue**.
4. From the toolbox, drag a Label control onto the Page Footer and in the Properties window, set the properties as follows.

Property Name	Property Value
Size	6.5, 0.2 in
Location	0, 0 in
Alignment	Center
Font Bold	True
ForeColor	White
Text	984-242-0700, https://www.grapecity.com/activereportsnet, us.sales@grapecity.com

To add code to overlay the data report pages with the letterhead report

To write the code in Visual Basic.NET

- Add the ActiveReports viewer control to the Windows Form. Then, double-click the top of the Windows Form to create an event-handling method for the form's Load event. Add code to the handler to:
  - Set the viewer to display the rptData report document
  - Overlay rptLetterhead on rptData

The following example shows what the code for the method looks like.

Visual Basic.NET code. Paste INSIDE the Form Load event.

```
Dim rpt As New rptData()
rpt.Run()
Dim rpt2 As New rptLetterhead()
rpt2.Run()
Dim i As Integer
For i = 0 To rpt.Document.Pages.Count - 1
   rpt.Document.Pages(i).Overlay(rpt2.Document.Pages(0))
Next
```

```
Viewer1.Document = rpt.Document
```

To write the code in C#

- Add the ActiveReports viewer control to the Windows Form. Then, double-click the top of the Windows Form to create an event-handling method for the form's Load event. Add code to the handler to:
  - Set the viewer to display the rptData report document
  - Overlay rptLetterhead on rptData

The following example shows what the code for the method looks like.

C# code. Paste INSIDE the Form Load event.

```
rptData rpt = new rptData();
rpt.Run();
rptLetterhead rpt2 = new rptLetterhead();
rpt2.Run();
for(int i = 0; i < rpt.Document.Pages.Count; i++)
{
   rpt.Document.Pages[i].Overlay(rpt2.Document.Pages[0]);
}
viewer1.Document = rpt.Document;
```

To view the report

Open the report in the Viewer. See Windows Forms Viewer for further information.

Run-Time Layouts

ActiveReports objects and controls are completely accessible at run time. You can modify the properties of any of the report sections or controls to produce a dynamic report. The section Format event allows you to modify the properties of the section and its controls, including height, visibility, and other visual properties. The Format event is the only event in which you can modify the printable area of a section. Once this event has run, any changes to the section's height are not reflected in the report output.

This walkthrough illustrates how to create a report layout at run time based on user input.

> **Note**: Add controls dynamically in the **ReportStart** event, otherwise, results may be unpredictable. For more information on events, see the Section Report Events topic.

This walkthrough is split into the following activities:

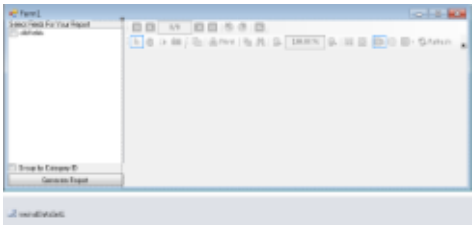
- Adding an ActiveReport to the Visual Studio project
- Adding controls to the Windows Form to display fields and a viewer
- Generating a dataset for the Windows Form
- Adding code to create the report layout
- Adding code to fill the check list with fields and to launch the report
- Adding code to alternate colors in the detail section
- Adding code to the ReportStart event to call the report layout code
- Adding code to the button's Click event to collect the selected values and launch the report

- Adding code to enable the button when fields are selected
- Adding code to the Form_Load event to call the fill check list code
- Viewing the report

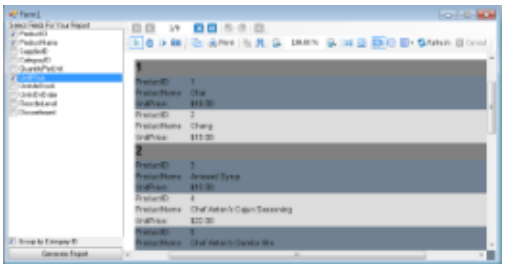
> **Note:** This walkthrough uses the NWind database. The NWIND.mdb file can be downloaded from GitHub: ..\Samples14\Data\NWIND.mdb.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout (Windows form)



Run-Time Layout



To add an ActiveReport to the Visual Studio project

1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 Section Report (code-based)** and in the Name field, rename the file as **rptRunTime**.
4. Click the **Add** button to open a new section report in the designer.

See Quick Start for information on adding different report layouts.

To add controls to the form

1. Resize the Windows Form so that it is large enough to accommodate a number of controls.
2. From the Visual Studio toolbox, drag the Panel control to the Windows Form and in the Properties Window, set the properties as follows.

   Panel

Property Name	Property Value
Dock	Left
Name	Panel1

3. From the Visual Studio toolbox, drag the following controls onto the Panel1 and in the Properties Window, set the

properties listed below.

Label

Property Name	Property Value
Dock	Top
Name	lblSelectFields
Text	Select Fields for Your Report

CheckedListBox

Property Name	Property Value
Dock	Fill
Name	clbFields

Button

Property Name	Property Value
Dock	Bottom
Name	btnGenRep
Text	Generate Report

CheckBox

Property Name	Property Value
Dock	Bottom
Name	chkGroup
Text	Group By Category ID

4. From the Visual Studio toolbox, drag the Viewer control to the Windows Form and in the Properties Window, set the properties as follows.

Viewer

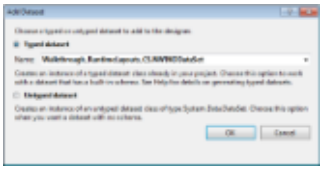
Property Name	Property Value
Dock	Fill
Name	Viewer1

To generate a dataset for the Form

1. From the **Project** menu, select **Add New Item**.
2. Select **DataSet**, rename the file NWINDDataSet.xsd and click the **Add** button.
3. In the DataSet Designer that appears, click the **Server Explorer** link.
4. In the Server Explorer, expand the node for your local copy of the Northwind database, then the **Tables** node, and drag the **Products** table onto the DataSet designer.

> **Tip:** If you do not see a copy of the Northwind database, click the **Connect to Database** icon on top of the Server Explorer and follow the prompts.

5. Open the Form and from the Visual Studio toolbox, drag DataSet onto the Form.
6. From the Add DataSet dialog that appears, select Typed dataset and click OK.



To add code to create the report layout

1. Right-click on **rptRunTime** and select **View Code**.
2. Add the following code within the class declaration of the report to:
   - Create an array of fields
   - Create an option for whether to use groups
   - Set properties on the report sections
   - Add textboxes and labels to the report based on the array of fields
   - Handle exceptions

To write the code in Visual Basic.NET

The following example shows what the code for the method looks like.

Visual Basic.NET code. Paste JUST BELOW the statements at the top of the code view

```
Imports GrapeCity.ActiveReports.SectionReportModel
```

Visual Basic.NET code. Paste INSIDE the class declaration of the report.

```
Private m_arrayFields As ArrayList
Private m_useGroups As Boolean
'Create an array to hold the fields selected by the user
Public WriteOnly Property FieldsList() As ArrayList
   Set(ByVal Value As ArrayList)
       m_arrayFields = Value
   End Set
End Property
'Create a property to hold the user's grouping choice
Public WriteOnly Property UseGroups() As Boolean
   Set(ByVal Value As Boolean)
       m_useGroups = False
       m_useGroups = Value
   End Set
End Property
Private m_defaultHeight As Single = 0.2F
Private m_defaultWidth As Single = 4.0F
Private m_currentY As Single = 0.0F
'Set up report formatting and add fields based on user choices
Private Sub constructReport()
   Try
```

```
      Me.Detail1.CanGrow = True
      Me.Detail1.CanShrink = True
      Me.Detail1.KeepTogether = True
      If m_useGroups = True Then
         'If the user wants grouping, add a group header and footer and set the grouping
field
          Me.Sections.InsertGroupHF()
          CType(Me.Sections("GroupHeader1"), GroupHeader).DataField = "CategoryID"
          Me.Sections("GroupHeader1").BackColor = System.Drawing.Color.Gray
          Me.Sections("GroupHeader1").CanGrow = True
          Me.Sections("GroupHeader1").CanShrink = True
          CType(Me.Sections("GroupHeader1"), GroupHeader).RepeatStyle =
RepeatStyle.OnPageIncludeNoDetail
          'Add a textbox to display the group's category ID
          Dim txt As New TextBox
          txt.DataField = "CategoryID"
          txt.Location = New System.Drawing.PointF(0.0F, 0)
          txt.Width = 2.0F
          txt.Height = 0.3F
          txt.Style = "font-weight: bold; font-size: 16pt"
          Me.Sections("GroupHeader1").Controls.Add(txt)
      End If
      Dim i As Integer
      For i = 0 To m_arrayFields.Count - 1
          'For all fields selected by the user (except CategoryID) create a label and a
textbox
          If m_arrayFields(i).ToString <> "CategoryID" Then
              Dim lbl As New Label
              'Set the label to display the name of the selected field
              lbl.Text = m_arrayFields(i) + ":"
              'Set the location of each label
               '(m_currentY gets the height of each control added on each iteration)
              lbl.Location() = New System.Drawing.PointF(0.0F, m_currentY)
              lbl.Width = 0.9F
              lbl.Height = m_defaultHeight
              Me.Detail1.Controls.Add(lbl)
              Dim txt As New TextBox
               'Set the textbox to display data
              txt.DataField = m_arrayFields(i)
               'Set the location of the textbox
              txt.Location = New System.Drawing.PointF(1.0F, m_currentY)
              txt.Width = m_defaultWidth
              txt.Height = m_defaultHeight
              Me.Detail1.Controls.Add(txt)
               'Set the textbox to use currency formatting if the field is UnitPrice
              If m_arrayFields(i) = "UnitPrice" Then
                 txt.OutputFormat = "$#.00"
              End If
               'Increment the vertical location by adding the height of the added controls
              m_currentY = m_currentY + m_defaultHeight
```

```
            End If
        Next
    Catch ex As Exception
        System.Windows.Forms.MessageBox.Show("Error in Report-constructReport: " +
ex.Message, "Project Error", System.Windows.Forms.MessageBoxButtons.OK,
System.Windows.Forms.MessageBoxIcon.Error)
    End Try
End Sub
```

To write the code in C#

The following example shows what the code for the method looks like.

C# code. Paste JUST BELOW the statements at the top of the code view

```
using GrapeCity.ActiveReports.SectionReportModel;
```

C# code. Paste INSIDE the class declaration of the report.

```
private ArrayList m_arrayFields;
 //Create an array to hold the fields selected by the user
public ArrayList FieldsList
{
   set{m_arrayFields = value;}
}
private bool m_useGroups = false;
 //Create a property to hold the user's grouping choice
public bool UseGroups
{
   set{m_useGroups = value;}
}
float m_defaultHeight = .2f;
float m_defaultWidth = 4f;
float m_currentY = 0f;
 //Set up report formatting and add fields based on user choices
private void constructReport()
{
   try
   {
      this.detail.CanGrow = true;
      this.detail.CanShrink = true;
      this.detail.KeepTogether = true;
      if(m_useGroups)
      {
         //If the user wants grouping, add a group header and footer and set the grouping
field
        this.Sections.InsertGroupHF();
        ((GroupHeader)this.Sections["GroupHeader1"]).DataField = "CategoryID";
        this.Sections["GroupHeader1"].BackColor = System.Drawing.Color.Gray;
        this.Sections["GroupHeader1"].CanGrow = true;
        this.Sections["GroupHeader1"].CanShrink = true;
        ((GroupHeader)this.Sections["GroupHeader1"]).RepeatStyle =
```

```
RepeatStyle.OnPageIncludeNoDetail;
        this.Sections["GroupFooter1"].Height = 0;
         //Add a textbox to display the group's category ID
        TextBox txt = new TextBox();
        txt.DataField = "CategoryID";
        txt.Location = new System.Drawing.PointF(0f,0);
        txt.Width =2f;
        txt.Height = .3f;
        txt.Style = "font-weight: bold; font-size: 16pt;";
        this.Sections["GroupHeader1"].Controls.Add(txt);
      }
      for(int i=0;i<m_arrayFields.Count;i++)
      {
        if(!m_useGroups || (m_useGroups && m_arrayFields[i].ToString() != "CategoryID"))
          //'For all fields selected by the user (except CategoryID) create a label and a
textbox
          {
           Label lbl = new Label();
            //Set the label to display the name of the selected field
           lbl.Text = m_arrayFields[i].ToString() + ":";
            //Set the location of each label
            //(m_currentY gets the height of each control added on each iteration)
           lbl.Location = new System.Drawing.PointF(0f,m_currentY);
           lbl.Width =.9f;
           lbl.Height = m_defaultHeight;
           this.detail.Controls.Add(lbl);
           TextBox txt = new TextBox();
            //Set the textbox to display data
           txt.DataField = m_arrayFields[i].ToString();
            //Set the location of the textbox
           txt.Location = new System.Drawing.PointF(1f,m_currentY);
           txt.Width = m_defaultWidth;
           txt.Height = m_defaultHeight;
           this.detail.Controls.Add(txt);
           //Set the textbox to use currency formatting if the field is UnitPrice
           if (m_arrayFields[i].ToString().Equals("UnitPrice"))
          {
              txt.OutputFormat = "$#.00";
          }
            //Increment the vertical location by adding the height of the added controls
           m_currentY = m_currentY + m_defaultHeight;
        }
      }
   }
   catch(Exception ex)
   {
      System.Windows.Forms.MessageBox.Show("Error in Report-constructReport: " +
ex.Message,"Project
Error",System.Windows.Forms.MessageBoxButtons.OK,System.Windows.Forms.MessageBoxIcon.Error);
   }
```

```
}
```

To add code to fill the check list with fields and to launch the report

1. Right-click the Windows Form and select **View Code**.
2. Add code within the class declaration of the form to:
    - Fill the check list with fields
    - Launch the report

To write the code in Visual Basic.NET

The following example shows what the code for the method looks like.

Visual Basic.NET code. Paste JUST BELOW the statements at the top of the code view

```
Imports System.Collections
```

Visual Basic.NET code. Paste INSIDE the class declaration of the form.

```
Dim i As Integer
Dim c As Integer
Dim m_arrayField As New ArrayList()
Private Sub fillCheckBox()
   For i = 0 To Me.NwindDataSet1.Tables.Count - 1
      For c = 0 To Me.NwindDataSet1.Tables(i).Columns.Count - 1
        Me.clbFields.Items.Add(Me.NwindDataSet1.Tables(i).Columns(c).ColumnName)
      Next
   Next
End Sub
Private Sub launchReport()
   Dim rpt As New rptRunTime()
   Dim dataAdapter As New NWINDDataSetTableAdapters.ProductsTableAdapter
      Try
           rpt.FieldsList = m_arrayField
           rpt.UseGroups = chkGroup.Checked
           dataAdapter.Fill(NwindDataSet1.Products)
           rpt.DataSource = Me.NwindDataSet1.Products
           Viewer1.Document = rpt.Document
           rpt.Run()
        Catch ex As Exception
           System.Windows.Forms.MessageBox.Show(Me, "Error in launchReport: " + ex.Message,
"Project Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
      End Try
End Sub
```

To write the code in C#

The following example shows what the code for the method looks like.

C# code. Paste JUST BELOW the statements at the top of the code view

```
using System.Collections;
```

C# code. Paste INSIDE the class declaration of the form.

```
ArrayList m_arrayField = new ArrayList();
private void fillCheckBox()
{
   for(int i = 0; i < this.nwindDataSet1.Tables.Count; i++)
   {
      for(int c = 0; c < this.nwindDataSet1.Tables[i].Columns.Count; c++)
      {
        this.clbFields.Items.Add(this.nwindDataSet1.Tables[i].Columns[c].ColumnName);
      }
   }
}
private void launchReport()
{
   try
   {
      rptRunTime rpt = new rptRunTime();
      rpt.FieldsList = m_arrayField;
      rpt.UseGroups = chkGroup.Checked;
      NWINDDataSetTableAdapters.ProductsTableAdapter dataAdapter = new
NWINDDataSetTableAdapters.ProductsTableAdapter();
      dataAdapter.Fill(this.nwindDataSet1.Products);
      rpt.DataSource = this.nwindDataSet1.Products;
      this.Viewer1.Document = rpt.Document;
      rpt.Run();
   }
   catch(Exception ex)
   {
      MessageBox.Show(this,"Error in launchReport: " + ex.Message,"Project
Error",MessageBoxButtons.OK,MessageBoxIcon.Error);
   }
}
```

To add code to alternate colors in the detail section

1. Double-click the detail section of rptRunTime. This creates an event-handling method for rptRunTime's Detail_Format event.
2. Add code to the handler to alternate colors for a green bar report effect.

To write the code in Visual Basic.NET

The following example shows what the code for the method looks like.

Visual Basic.NET code. Paste JUST ABOVE the Detail Format event.

```
Dim m_count As Integer
```

Visual Basic.NET code. Paste INSIDE the Detail Format event.

```
If m_count Mod 2 = 0 Then
  Me.Detail1.BackColor = System.Drawing.Color.SlateGray
Else
  Me.Detail1.BackColor = System.Drawing.Color.Gainsboro
```

```
End If
m_count = m_count + 1
```

To write the code in C#

The following example shows what the code for the method looks like.

C# code. Paste JUST ABOVE the Detail Format event.

```
int m_count;
```

C# code. Paste INSIDE the Detail Format event.

```
if(m_count % 2 == 0)
{
  this.detail.BackColor = System.Drawing.Color.SlateGray;
}
else
{
  this.detail.BackColor = System.Drawing.Color.Gainsboro;
}
  m_count++;
```

To add code to the ReportStart event to call the report layout code

1. Double-click the gray area below rptRunTime to create an event-handling method for rptRunTime's ReportStart event.
2. Add code to call the constructReport method.

To write the code in Visual Basic.NET

The following example shows what the code for the method looks like.

Visual Basic.NET code. Paste INSIDE the ReportStart event.

```
constructReport()
```

To write the code in C#

The following example shows what the code for the method looks like.

C# code. Paste INSIDE the ReportStart event.

```
constructReport();
```

To add code to the button's Click event to collect the selected values and launch the report

1. Double-click **btnGenRep** to create an event-handling method for the button click event.
2. Add code to the handler to collect the selected values and launch the report.

To write the code in Visual Basic.NET

The following example shows what the code for the method looks like.

Visual Basic.NET code. Paste INSIDE the button click event.

```
Me.m_arrayField.Clear()
```

```
For Me.i = 0 To Me.clbFields.CheckedItems.Count - 1
   m_arrayField.Add(Me.clbFields.CheckedItems(i).ToString)
Next
launchReport()
```

To write the code in C#

The following example shows what the code for the method looks like.

C# code. Paste INSIDE the button click event.

```
this.m_arrayField.Clear();
for(int i = 0; i < this.clbFields.CheckedItems.Count; i++)
{
  m_arrayField.Add(this.clbFields.CheckedItems[i].ToString());
}
launchReport();
```

To add code to enable the button when fields are selected

1. Select the checked list box (**clbFields**) and go to the Properties Window.
2. At the top of Properties Window, select the **Events** icon to open the events list.
3. Double-click the **SelectedIndexChanged** event. This creates an event-handling method for the clbFields_SelectedIndexChanged event.
4. Add code to the handler to enable the button when fields are selected.

The following example shows what the code for the method looks like.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the SelectedIndexChanged event.

```
If Me.clbFields.CheckedItems.Count < 0 Then
    Me.btnGenRep.Enabled = False
Else
    Me.btnGenRep.Enabled = True
End If
```

To write the code in C#

C# code. Paste INSIDE the SelectedIndexChanged event.

```
if(this.clbFields.CheckedItems.Count>0)
{
  this.btnGenRep.Enabled = true;
}
else
{
  this.btnGenRep.Enabled = false;
}
```

To add code to the Form_Load event to call the fill check list code

1. Double-click the title bar of the form. This creates an event-handling method for the Windows Form_Load event.
2. Add code to the handler to call the fillCheckBox() method to populate clbFields with field values and to handle exceptions.

To write the code in Visual Basic.NET

The following example shows what the code for the method looks like.

Visual Basic.NET code. Paste INSIDE the Form Load event.

```
Try
  fillCheckBox()
Catch ex As Exception
  System.Windows.Forms.MessageBox.Show(Me, "Error in Form1_Load: " + ex.Message, "Project
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try
```

To write the code in C#

The following example shows what the code for the method looks like.

C# code. Paste INSIDE the Form Load event.

```
try
{
  fillCheckBox();
}
catch(Exception ex)
{
  MessageBox.Show(this,"Error in Form1_Load: " + ex.Message,"Project Error",
MessageBoxButtons.OK,MessageBoxIcon.Error);
}
```

To view the report

- Press F5 to run the project.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

Subreports with XML Data

Using XML data requires some setup that is different from other types of data. This walkthrough illustrates how to set up a subreport bound to the XML DataSource in the parent report.

This walkthrough is split up into the following activities:

- Adding an ActiveReport to the Visual Studio project
- Connecting the parent report to an XML data source
- Adding controls to display the data
- Adding code to create a new instance of the subreport

- Adding code to pass a subset of the parent report's data to the subreport
- Viewing the report

> **Note:** This walkthrough uses Customer.xml. The Customer.xml file can be downloaded from GitHub: ..\Samples14\Data\NWIND.mdb.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout

Run-Time Layout

To add an ActiveReport to the Visual Studio project

1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 Section Report (code-based)** and in the Name field, rename the file as **rptMain**.
4. Click the **Add** button to open a new section report in the designer.
5. From the **Project** menu, select **Add New Item**.
6. In the Add New Item dialog that appears, select **ActiveReports 14 Section Report (code-based)** and in the Name field, rename the file as **rptSub**.
7. Click the **Add** button to open a second new section report in the designer.

See Quick Start for information on adding different report layouts.

To connect the Parent Report (rptMain) to a data source

1. On the detail section band, click the Data Source Icon.

2. In the Report Data Source dialog, on the **XML** tab, click the ellipsis (...) button next to File URL field.
3. In the **Open File** window that appears, navigate to **Customer.xml** and click the **Open** button.
4. In the **Recordset Pattern** field, enter `//CUSTOMER`.
5. Click **OK** to save the data source and return to the report design surface.

To create a layout for the Parent Report (rptMain)

1. On the design surface, select the pageHeader section and in the Properties window, set the **Height** property to **0.3**.
2. On the design surface, select the grey area outside the report and in the Properties window, set the

PrintWidth property to **6.5**.

3. On the design surface, select the detail section and in the Properties window, set the **CanShrink** property to **True** to eliminate white space.
4. From the toolbox, drag the Label control onto the pageHeader section and in the Properties window, set the properties as follows.

Property Name	Property Value
Text	Orders by Customer
Location	0, 0 in
Size	6.5, 0.25 in
Font	Arial, 14pt, style=Bold
Alignment	Center

5. From the toolbox, drag the controls onto the detail section and in the Properties window, set the properties of each control as follows.

TextBox1

Property Name	Property Value
DataField	NAME
Location	1.2, 0 in
Size	2, 0.2 in

Label1

Property Name	Property Value
Text	Customer Name:
Location	0, 0 in
Size	1.2, 0.2 in
Font Bold	True

Label2

Property Name	Property Value
Text	Orders:
Location	1.2, 0.25 in
Size	1, 0.2 in
Font Bold	True

Subreport

Property Name	Property Value

Location	2.3, 0.25 in
Size	4, 1 in

To create a layout for the Child Report (rptSub)

1. On the design surface, select the detail section and in the Properties window, set the properties as follows.

Property Name	Property Value
CanShrink	True
BackColor	LightSteelBlue

> **Tip**: Even if you do not want colors in your finished reports, using background colors on subreports can help in troubleshooting layout issues.

2. On the design surface, right-click the pageHeader or pageFooter section and select **Delete**. Subreports do not render these sections, so deleting them saves processing time.
3. From the toolbox, drag the following controls to the detail section and in the Properties window, set the properties as follows.

TextBox1

Property Name	Property Value
DataField	TITLE
Name	txtTitle
Location	0, 0 in
Size	2.9, 0.2 in

TextBox2

Property Name	Property Value
DataField	PRICE
Name	txtPrice
Location	3, 0 in
Size	1, 0.2 in
Alignment	Right
OutputFormat	$#,##0.00 (or select Currency in the dialog)

To add code to create a new instance of the Child Report (rptSub)

> **Warning**: Do not create a new instance of the subreport in the **Format** event. Doing so creates a new subreport each time the section Format code is run, which uses a lot of memory.

To write the code in Visual Basic

1. Right-click the design surface of **rptMain** and select **View Code**.
2. At the top left of the code view of the report, click the drop-down arrow and select **(rptMain Events)**.
3. At the top right of the code window, click the drop-down arrow and select **ReportStart**. This creates an event-handling method for the ReportStart event.
4. Add code to the handler to create an instance of rptSub.

The following example shows what the code for the method looks like.

Visual Basic.NET code. Paste JUST ABOVE the ReportStart event.

```
Dim rpt As rptSub
```

Visual Basic.NET code. Paste INSIDE the ReportStart event.

```
rpt = New rptSub
```

To write the code in C#

1. Click in the gray area below **rptMain** to select it.
2. Click the events icon in the Properties Window to display available events for the report.
3. Double-click **ReportStart**. This creates an event-handling method for the report's ReportStart event.
4. Add code to the handler to create a new instance of rptSub.

The following example shows what the code for the method looks like.

C# code. Paste JUST ABOVE the ReportStart event.

```
private rptSub rpt;
```

C# code. Paste INSIDE the ReportStart event.

```
rpt = new rptSub();
```

To add code to pass a subset of the Parent Report's data to the Child Report

To add code to pass a subset of the parent report's data to the subreport

1. Double-click in the detail section of the design surface of rptMain to create a detail_Format event.
2. Add code to the handler to:
    - Create a new GrapeCity XMLDataSource
    - Type cast the new data source as rptMain's data source and set the NodeList to the "ORDER/ITEM" field
    - Display rptSub in the subreport control
    - Pass the new data source to the subreport

To write the code in Visual Basic

The following example shows what the code for the method looks like.

Visual Basic.NET code. Paste INSIDE the Format event.

```
    Dim xmlDS As New GrapeCity.ActiveReports.Data.XMLDataSource
    xmlDS.NodeList = CType(CType(Me.DataSource,
GrapeCity.ActiveReports.Data.XMLDataSource).Field("ORDER/ITEM", True),
System.Xml.XmlNodeList)
    rpt.DataSource = xmlDS
```

```
    SubReport1.Report = rpt
```

To write the code in C#

The following example shows what the code for the method looks like.

C# code. Paste INSIDE the Format event.

```
   GrapeCity.ActiveReports.Data.XMLDataSource xmlDS = new
GrapeCity.ActiveReports.Data.XMLDataSource();
   xmlDS.NodeList = (System.Xml.XmlNodeList)
((GrapeCity.ActiveReports.Data.XMLDataSource) this.DataSource).Field("ORDER/ITEM",
true);
   rpt.DataSource = xmlDS;
   subReport1.Report = rpt;
```

To view the report

- Click the preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

Subreports with Run-Time Data Sources

ActiveReports allows section reports to contain any number of child reports using the Subreport control. Child reports, or subreports, are executed each time the parent section (i.e. the section in which the Subreport control is placed) is processed. This walkthrough illustrates how to modify the subreport record source from the data in the parent report to retrieve the correct information.

This walkthrough is split up into the following activities:

- Adding a main report and a subreport to a Visual Studio project
- Connecting the main report to a data source
- Adding controls to the main report to display data and contain the subreport
- Adding controls to the subreport to display data
- Adding code to save the current record's CategoryID for use in the subreport's SQL query
- Adding code to create an instance of the subreport
- Adding code to assign a data source for the subreport
- Viewing the report

> **Note:** This walkthrough uses tables from the NWind database. The NWIND.mdb file can be downloaded from GitHub: ..\Samples14\Data\NWIND.mdb.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.

Design-Time Layout

Run-Time Layout

To add an ActiveReport to the Visual Studio project

1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 Section Report (code-based)** and in the Name field, rename the file as **rptMain**.
4. Click the **Add** button to open a new section report in the designer.
5. From the **Project** menu, select **Add New Item**.
6. In the Add New Item dialog that appears, select **ActiveReports 14 Section Report (code-based)** and in the Name field, rename the file as **rptSub**.
7. Click the **Add** button to open a second new section report in the designer.

See Quick Start for information on adding different report layouts.

To connect the Parent Report (rptMain) to a data source

1. On the detail section band, click the Data Source Icon.
   

2. In the Report Data Source dialog that appears, from the **OLE DB** tab, create a data source connection. See Bind Reports to a Data Source for further details.
3. Once the connection string field is populated, in the Query field, enter the following SQL query.

SQL Query
`SELECT * FROM Categories`

4. Click **OK** to save the data source and return to the report design surface.

To create a layout for the Parent Report (rptMain)

1. In the Report Explorer, select the report and in the Properties window, set the **PrintWidth** property to **5.75**.
2. On the design surface, select the detail section and in the Properties window, set the **CanShrink** property to **True** to eliminate white space.
3. From the toolbox, drag a Label control onto the pageHeader section and in the Properties window, set the properties as follows.

Property Name	Property Value

Name	lblProductsbyCategory
Text	Products by Category
Location	0, 0 in
Size	5.75, 0.25 in
Font Size	14
Alignment	Center

4. From the toolbox, drag the following controls onto the detail section and in the Properties window, set the properties as follows.

TextBox1

Property Name	Property Value
Name	txtCategoryID1
DataField	CategoryID
Visible	False

TextBox2

Property Name	Property Value
Name	txtCategoryName1
DataField	CategoryName
Location	1.15, 0.05 in

Label1

Property Name	Property Value
Name	lblCategoryName
Text	CategoryName:
Location	0, 0.05 in
Size	1.15, 0.2 in
Font Bold	True

Label2

Property Name	Property Value
Name	lblProducts
Text	Products:
Location	2.4, 0.05 in

Font Bold	True

Subreport

Property Name	Property Value
Name	SubReport1
Location	3.5, 0.05 in
Size	2.25, 1 in

To create a layout for the Child Report (rptSub)

1. On the design surface, select the detail section and in the Properties window, set the following properties.

Property Name	Property Value
CanShrink	True
BackColor	AliceBlue

> **Tip**: Even if you do not want colors in your finished reports, using background colors on subreports can help in troubleshooting layout issues.

2. On the design surface, right-click the pageHeader or pageFooter section and select **Delete**. Subreports do not render these sections, so deleting them saves processing time.
3. From the toolbox, drag a TextBox control to the detail section and in the Properties window, set the following properties.

Property Name	Property Value
DataField	ProductName
Name	txtProductName
Text	Product Name
Location	0, 0 in
Size	2.25, 0.2 in

To add code to create an instance of the subreport

> ⚠ **Warning**: Do not create a new instance of the subreport in the **Format** event. Doing so creates a new subreport each time the section Format code is run, which uses a lot of memory.

To write the code in Visual Basic

1. At the top left of the code view for the report, click the drop-down arrow and select **(rptMain Events)**.
2. At the top right of the code window, click the drop-down arrow and select **ReportStart**. This creates an event-handling method for the report's ReportStart event.
3. Add code to the handler to create a new instance of the subreport.

The following example shows what the code for the method looks like.

Visual Basic.NET code. Paste JUST ABOVE the ReportStart event.

```
Private rpt As rptSub
Private childDataSource As New GrapeCity.ActiveReports.Data.OleDBDataSource()
```

Visual Basic.NET code. Paste INSIDE the ReportStart event.

```
rpt = New rptSub()
```

To write the code in C#

1. Click in the gray area below rptMain to select it.
2. Click the events icon in the Properties Window to display available events for the report.
3. Double-click **ReportStart**. This creates an event-handling method for the report's ReportStart event.
4. Add code to the handler to create a new instance of the subreport.

The following example shows what the code for the method looks like.

C# code. Paste JUST ABOVE the ReportStart event.

```
private rptSub rpt;
private GrapeCity.ActiveReports.Data.OleDBDataSource childDataSource = new
GrapeCity.ActiveReports.Data.OleDBDataSource();
```

C# code. Paste INSIDE the ReportStart event.

```
rpt = new rptSub();
```

To add code to assign a data source for the Child Report (rptSub)

1. Back in design view of the Parent report (rptMain), double-click the detail section. This creates the Detail_Format event handler.
2. Add code to the handler to:
    - Set the connection string for the OleDBDataSource for the subreport
    - Set the SQL query for the new data source and pass in the current record's CategoryID
    - Set the data source of the subreport to the data source
    - Assign rptSub to the SubReport control

To write the code in Visual Basic

The following example shows what the code for the method looks like.

Visual Basic.NET code. Paste INSIDE the Format event.

```
childDataSource.ConnectionString = CType(Me.DataSource,
GrapeCity.ActiveReports.Data.OleDBDataSource).ConnectionString
childDataSource.SQL = "SELECT * FROM Products WHERE CategoryID = " +
Me.txtCategoryID1.Value.ToString
rpt.DataSource = childDataSource
SubReport1.Report = rpt
```

To write the code in C#

C# code. Paste INSIDE the Format event.

```
childDataSource.ConnectionString =
((GrapeCity.ActiveReports.Data.OleDBDataSource)this.DataSource).ConnectionString;
childDataSource.SQL = "SELECT * FROM Products WHERE CategoryID = " +
this.txtCategoryID1.Value.ToString();
rpt.DataSource = childDataSource;
SubReport1.Report = rpt;
```

To view the report

- Click the preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

Chart

This section contains the following walkthroughs that fall under the Chart category.

Bar Chart
: This walkthrough demonstrates how to create a bar chart which compares items across categories.

3D Pie Chart
: This walkthrough demonstrates how to a three dimensional pie chart which shows how the percentage of each data item contributes to a total percentage.

Financial Chart
: This walkthrough demonstrates how to create a financial chart which lets you plot high, low, opening, and closing prices.

Unbound Chart
: This walkthrough demonstrates how to create a simple unbound chart.

Bar Chart

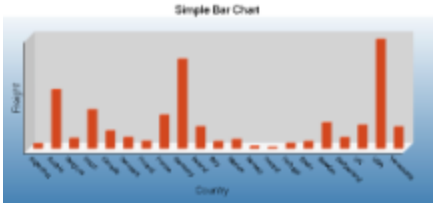
Bar charts are useful in comparing items across categories. This walkthrough illustrates how to create a simple bar chart using the ActiveReports chart control.

The walkthrough is split up into the following activities:

- Adding a chart control to the report
- Setting a data source for the chart
- Setting the chart's properties

> **Note:** This walkthrough uses the Northwind database. The NWIND.mdb file can be downloaded from GitHub: ..\Samples14\Data\NWIND.mdb.

When you complete this walkthrough you get a layout that looks similar to the following at run time.



To add an ActiveReport to the Visual Studio project

1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 Section Report (code-based)** and in the Name field, rename the file as **BarChart**.
4. Click the **Add** button to open a new section report in the designer.

See Quick Start for information on adding different report layouts.

To add the Chart control to the report

1. From the toolbox, drag the **Chart** data region to the body of the report.
2. If the chart wizard appears, click **Cancel**.

   > **Tip:** If you do not want the chart wizard to appear each time you add a chart, clear the **Auto Run Wizard** checkbox. You can still access the wizard via the command verbs (see below).

3. On the design surface, select the grey area outside the report and in the Properties window, set the **PrintWidth** property to **6.5**.
4. In the Properties window, set the following properties.

Property Name	Property Value
Location	0, 0 in
Size	6.5, 3.5 in

5. In the Report Explorer, select **Detail1** and go to the properties window to set the **Height** property to **3.5**.

To connect the Chart to a data source

1. Select the Chart control and at the bottom of the Properties window, select the **Data Source** command. See Properties Window for further details on accessing commands.

   > **Tip**: If the verb is not visible, right-click an empty space in the Properties Window and select **Commands** to display verbs.

2. In the Chart DataSource dialog box that appears, click the **Build** button.
3. In the Data Link Properties window, select **Microsoft Jet 4.0 OLE DB Provider** and click the **Next** button.
4. Click the ellipsis button (...) to browse to the Northwind database. Click **Open** once you have selected the file.
5. Click the **OK** button to close the window and fill in the Connection String.
6. In the **Query** field, enter the following SQL query.

SQL Query
`SELECT ShipCountry, SUM(Freight) AS FreightSum FROM Orders GROUP BY ShipCountry`

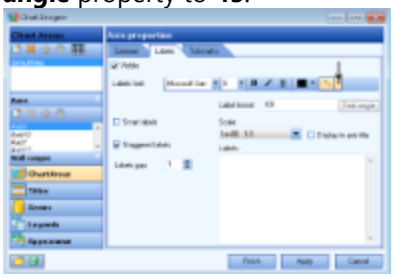
7. Click **OK** to save the data source.

To configure the appearance of the Chart

1. Select the Chart control and at the bottom of the Properties window, select the **Customize** command. See Properties Window for further details on accessing commands.
2. In the **Chart Designer** dialog that appears set the following.

   Chart Areas

   1. Click the **Axes** bar on the left to expand it.
   2. Click **Axis X**, and on the **Common** tab in the pane to the right, type **Country** in the **Title** textbox and set the **Font size** to **12**.
   3. On the **Labels** tab, select the **Staggered Labels** checkbox to avoid overlapping labels and set the **Text angle** property to **45**.

   4. Click **Axis Y** on the left, and on the **Common** tab in the pane to the right, type **Freight** in the **Title** textbox and set the **Font size** to **12**.

   Titles

   1. Click the **Titles** bar on the left to expand it. In the list of titles, the **header** is selected by default.
   2. In the **Caption** textbox, type **Simple Bar Chart** and increase the **Font size** to **14**.
   3. In the list of titles to the left, select the footer and delete it by clicking the **Delete** icon on top of the list.

   Series

   1. Click the **Series** bar on the left. The **Series1** is selected by default.
   2. In the **Data Binding** box, set **X (Name)** to **ShipCountry**, and set **Y** to **FreightSum**.
   3. In the list of series to the left, select Series2 and Series3 and delete them by clicking the **Delete** icon on top of the list.

   Legend

   1. Click the **Legend** bar on the left to expand it. The **defaultLegend** is selected by default.
   2. On the **Common** tab, clear the **Visible** checkbox to hide the legend.
3. Click **Finish** to exit the **Chart Designer**.

To view the report

- Click the preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

3D Pie Chart

Pie charts are useful in showing how the percentage of each data item contributes to the total. This walkthrough illustrates

how to create a three dimensional pie chart.

The walkthrough is split up into the following activities:

- Adding a chart control to the report
- Adding a series and data points to the chart
- Setting the chart's properties

When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.

To add an ActiveReport to the Visual Studio project

1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 Section Report (code-based)** and in the Name field, rename the file as **3DPieChart**.
4. Click the **Add** button to open a new section report in the designer.

See Quick Start for information on adding different report layouts.

To add the Chart control to the report

1. From the toolbox, drag the **Chart** data region to the body of the report.
2. If the chart wizard appears, click **Cancel**.

   > **Tip:** If you do not want the chart wizard to appear each time you add a chart, clear the **Auto Run Wizard** checkbox. You can still access the wizard via the command verbs (see below).

3. In the Properties window, set the following properties.

Property Name	Property Value
Location	0, 0in
Size	6.5, 3.5in

4. In the Report Explorer, select **Detail1** and go to the properties window to set the **Height** property to **3.5**.
5. On the design surface, select the grey area outside the report and in the Properties window, set the **PrintWidth** property to **6.5**.

To add a series and data points to the Chart

1. With the chart control selected, go to the Properties window and click the **Series (Collection)** property , then click the ellipsis button (...) that appears.
2. In the **Series Collection Editor** that appears, **Series1** is selected by default. There, under Series1 properties, change the following.

Property Name	Property Value
ColorPalette	Confetti
Type	Doughnut3D

3. Click the **Points (Collection)** property, then click the ellipsis button that appears.
4. In the **DataPoint Collection** that appears, click the **Add** button to add a data point.
5. In the **DataPoint Collection Editor** that appears, go to the Properties window to set the following properties.

Property Name	Property Value
LegendText	Figs
YValues	19
Properties>ExplodeFactor	0.5

6. Click the **Add** button to add another data point.
7. In the **DataPoint Collection Editor** that appears, go to the Properties window to set the following properties.

Property Name	Property Value
LegendText	Raspberries
YValues	15

8. Click the **Add** button to add another data point.
9. In the **DataPoint Collection Editor** that appears, go to the Properties window to set the following properties.

Property Name	Property Value
LegendText	Blueberries
YValues	37

10. Click the **Add** button to add another data point.
11. In the **DataPoint Collection Editor** that appears, go to the Properties window to set the following properties.

Property Name	Property Value
LegendText	Bananas
YValues	21

12. Click **OK** to save the data points and return to the Series Collection Editor.
13. In the **Series Collection Editor** under **Members**, select **Series2** and **Series3** and click the **Remove** button.
14. Click **OK** to save the changes and return to the report design surface.

To configure the appearance of the Chart

1. With the chart control selected, go to the Properties window and click the **ChartAreas (Collection)** property and then click the ellipsis button that appears.
2. In the **ChartArea Collection Editor** that appears, expand the **Projection** property node and set the **VerticalRotation** property to **50**. This allows you to see more of the top of the pie.
3. Click **OK** to return to the report design surface.
4. With the chart control highlighted, go to the Properties window and click the **Titles (Collection)** property and then click the ellipsis button that appears.
5. In the **Title Collection Editor** that appears, under header properties, set the following properties.

Property Name	Property Value
Text	3D Pie Chart
Font Size	14

6. Under **Members**, select the footer and click the **Remove** button.
7. Click **OK** to return to the report design surface.

To view the report

- Click the preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

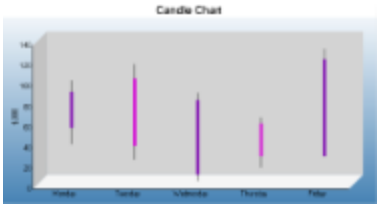
Financial Chart

Financial charts are useful for displaying stock information using High, Low, Open and Close values. This walkthrough illustrates how to create a Candle chart.

The walkthrough is split up into the following activities:

- Adding a chart control to the report
- Adding a series and data points to the chart
- Setting the chart's properties

When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.



To add an ActiveReport to the Visual Studio project

1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 Section Report (code-based)** and in the Name field, rename the file as **FinancialChart**.
4. Click the **Add** button to open a new section report in the designer.

See Quick Start for information on adding different report layouts.

To add the Chart control to the report

1. From the toolbox, drag the **Chart** data region to the body of the report.
2. If the chart wizard appears, click **Cancel**.

   > **Tip:** If you do not want the chart wizard to appear each time you add a chart, clear the **Auto Run Wizard** checkbox. You can still access the wizard via the command verbs (see below).

3. In the Properties window, set the following properties.

Property Name	Property Value
Location	0, 0in
Size	6.5, 3.5in

4. In the Report Explorer, select **Detail1** and go to the properties window to set the **Height** property to **3.5**.On the design surface, select the grey area outside the report and in the Properties window, set the **PrintWidth** property

to **6.5**.

To add a series and data points to the Chart

1. With the chart control selected, go to the Properties window and click the **Series** (Collection) property and then click the ellipsis button.
2. In the **Series Collection Editor** that appears, **Series1** is selected by default. There, under Series1 properties, change the following.

Property Name	Property Value
Type	Candle
Properties>BodyDownswingBackdrop	(Default)
Properties>BodyDownswingBackdrop>Color	Fuchsia
Properties>BodyUpswingBackdrop	(Default)
Properties>BodyUpswingBackdrop>Color	DarkViolet
Properties>BodyWidth	5
Properties>WickLine	(Default)
Legend	(none)

3. Click the **Points** (Collection) property, then click the ellipsis button that appears.
4. In the DataPoint Collection window that appears, click **Add** to add a data point and set its **YValues** property to **99; 37; 53; 88**.

   > **Note**: The first Y value is the high figure or top of the wick; the second is the low figure, or bottom of the wick; the third is the opening figure; the fourth is the closing figure. If the fourth figure is higher than the third, the candle is DarkViolet, the BodyUpswingBackdrop.

5. Click **Add** to add another data point and set its **YValues** property to **115; 22; 101; 35**.
6. Click **Add** to add another data point, and set its **YValues** property to **87; 1; 7; 80**.
7. Click **Add** to add another data point, and set its **YValues** property to **63; 14; 57; 25**.
8. Click **Add** to add another data point, and set its **YValues** property to **130; 25; 25; 120**.
9. Click **OK** to save the data points and close the window.
10. In the **Series Collection Editor** under **Members**, select **Series2** and **Series3** and click the **Remove** button.
11. Click **OK** to return to the report design surface.

To configure the appearance of the Chart

1. With the chart control selected, go to the Properties window and click the **ChartAreas** (Collection) property and then click the ellipsis button that appears.
2. In the **ChartArea Collection Editor** that appears, under defaultArea properties, click the **Axes** (Collection) property, then click the ellipsis button that appears.
3. In the **AxisBase Collection Editor** that appears, set the following properties.

   AxisBase

   1. Under AxisX properties, in the **Title** property delete the default text.
   2. Click the **Labels** (Collection) property, then click the ellipsis button that appears. This is where you add the labels that appear along the X axis, the line across the bottom of the chart.
   3. In the **Array Data Editor** that appears, enter the following into the editor, each item on a separate line:
      - Monday

- Tuesday
- Wednesday
- Thursday
- Friday

4. Click the **OK** button to return to the **AxisBase Collection Editor**.
5. Under **Members**, select the **AxisY** member, and under AxisY properties set the following properties.

Property Name	Property Value
MajorTick>Step	10
LabelsVisible	True
Min	0
Title	$,000

6. Click **OK** to return to the **ChartArea Collection Editor**.

4. Click **OK** to return to the report design surface and see the changes reflected in the chart.
5. With the chart control selected, go to the Properties window and click the **Titles** (Collection) property and then click the ellipsis button that appears.
6. In the **Title Collection Editor** that appears, set the following properties.

Titles

1. Under **header properties**, set the **Text** property to **Candle Chart.**
2. Expand the **Font** property and set **Font Size** to **14**.
3. Under **Members**, select footer and click the **Remove** button.

7. Click **OK** to return to the report design surface.
8. With the Chart control selected, go to the Properties window and click the **Legends** (Collection) property and then click the ellipsis button that appears.
9. In the **Legend Collection Editor** that appears, set the following properties.

Legends

1. In the Properties window, set the **Visible** property to **False.**
2. Click **OK** to return to the report design surface and see the completed chart.

To view the report

- Click the preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

Unbound Chart

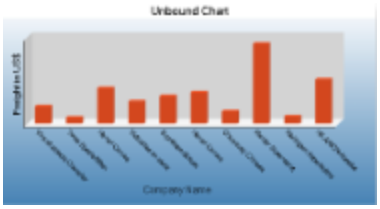
The Chart control allows you to bind charts to any type of data source, including arrays. You can create a chart without setting its data source and load the data into the control at run time. This walkthrough illustrates how to create a simple unbound chart.

The walkthrough is split up into the following activities:

- Adding the chart control to the report and setting chart properties
- Adding code to create the chart at run time

> **Note:** This walkthrough uses the Northwind database. The NWIND.mdb file can be downloaded from GitHub: ..\Samples14\Data\NWIND.mdb.

When you complete this walkthrough you get a layout that looks similar to the following at run time.



To add an ActiveReport to the Visual Studio project

1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 Section Report (code-based)** and in the Name field, rename the file as **UnboundChart**.
4. Click the **Add** button to open a new section report in the designer.

See Quick Start for information on adding different report layouts.

To add the Chart control to the report

1. From the toolbox, drag the **Chart** data region to the body of the report.
2. If the chart wizard appears, click **Cancel**.

   > **Tip:** If you do not want the chart wizard to appear each time you add a chart, clear the **Auto Run Wizard** checkbox. You can still access the wizard via the command verbs (see below).
3. In the Properties window, set the following properties.

Property Name	Property Value
Location	0, 0in
Size	6.5, 3.5in

4. In the Report Explorer, select **Detail** and go to the properties window to set the **Height** property to **3.5**.
5. On the design surface, select the grey area outside the report and in the Properties window, set the **PrintWidth** property to **6.5**.

To configure the appearance of the Chart

1. Select the Chart control and at the bottom of the Properties window, select the **Customize** command. See Properties Window for further details on accessing commands.

   > **Tip:** If the verb is not visible, right-click an empty space in the Properties Window and select **Commands** to display verbs.
2. In the **ChartAreas** view which displays by default, click the **Axes** bar to expand it.
3. Click **Axis X**, and on the **Common** tab in the pane to the right, type **Company Name** in the **Title** textbox and set the font size to **12**.

4. Click **Axis Y** on the left, and on the **Common** tab in the pane to the right, type **Freight in US$** in the **Title** textbox and increase the **Font size** to **12**.
5. Click the **Titles** bar on the left. In the list of titles, **header** is selected by default.
6. On the **Title properties** page, type **Unbound Chart** in the **Caption** textbox and set the **Font size** to **14**.
7. Under **Titles**, select the **footer** and delete it by clicking the **Delete** icon on top of the list.
8. Click the **Series** bar on the left.
9. Under Series, select **Series1**, **Series2** and **Series3** and delete them by clicking the **Delete** icon on top of the list.
10. Click the **Legends** bar on the left. The **defaultLegend** is selected by default.
11. On the **Common** page, clear the **Visible** checkbox to hide the legend.
12. Click the **Finish** button to exit the Chart Designer.

Back on the design surface of the report, the chart appears empty except for the title.

To add the code to create a chart at run time chart in Visual Basic or C#

Double-click the gray area below the report. This creates an event-handling method for rptUnboundChart's ReportStart event. Add code to the handler to:

- Create the series
- Create the dataset
- Set the chart properties
- Angle the labels to avoid overlap

The following examples show what the code for the methods look like in Visual Basic.NET and C#.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the ReportStart event.

```
'create the series
Dim series As New GrapeCity.ActiveReports.Chart.Series
series.Type = Chart.ChartType.Bar3D

'connection string and data adapter
Dim dbPath As String = "[User Folder]\Samples14\Data\NWIND.mdb"
Dim connString As String = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source= " + dbPath"
Dim da As New System.Data.OleDb.OleDbDataAdapter("SELECT * from Orders WHERE OrderDate <
#08/17/1994#", connString)

'create the dataset
Dim ds As New DataSet
da.Fill(ds, "Orders")

'set chart properties
Me.ChartControl1.DataSource = ds
Me.ChartControl1.Series.Add(series)
Me.ChartControl1.Series(0).ValueMembersY = ds.Tables("Orders").Columns(7).ColumnName
Me.ChartControl1.Series(0).ValueMemberX = ds.Tables("Orders").Columns(8).ColumnName

'angle the labels to avoid overlapping
Me.ChartControl1.ChartAreas(0).Axes(0).LabelFont.Angle = 45
```

To write the code in C#

C# code. Paste INSIDE the ReportStart event.

```
//create the series
GrapeCity.ActiveReports.Chart.Series series = new
GrapeCity.ActiveReports.Chart.Series();
series.Type = GrapeCity.ActiveReports.Chart.ChartType.Bar3D;

//connection string and data adapter
string dbPath = "[User Folder]\Samples14\Data\NWIND.mdb";
string connString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source= " + dbPath;
System.Data.OleDb.OleDbDataAdapter da = new System.Data.OleDb.OleDbDataAdapter
("SELECT * from Orders WHERE OrderDate < #08/17/1994#", connString);

// create the dataset
System.Data.DataSet ds = new System.Data.DataSet();
da.Fill(ds, "Orders");

// set chart properties
this.chartControl1.DataSource = ds;
this.chartControl1.Series.Add(series);
this.chartControl1.Series[0].ValueMembersY = ds.Tables["Orders"].Columns[7].ColumnName;
this.chartControl1.Series[0].ValueMemberX = ds.Tables["Orders"].Columns[8].ColumnName;

// angle the labels to avoid overlapping
this.chartControl1.ChartAreas[0].Axes[0].LabelFont.Angle = 45;
```

To view the report

- Click the preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

Export

This section contains the following walkthroughs that fall under the Export category.

Custom Web Exporting (Std Edition)
: This walkthrough demonstrates how to set up report custom exporting to PDF, Excel, TIFF, RTF, and plain text formats.

Custom HTML Outputter
: This walkthrough demonstrates how to create a custom HTML outputter for your ActiveReports ASP.NET Web Application.

Custom Web Exporting (Std Edition)

ActiveReports provides components that allow you to set up report custom exporting to PDF, Excel, TIFF, RTF, and plain text formats. You can similarly export to HTML, or you can create a custom HTML outputter.

To add a report and export references to the Web project

1. From the **View** menu, select **Component Designer** to go to the design view of the aspx file.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item window that appears, select the **ActiveReports 14 Section Report (xml-based)** template, set the report's name to **SectionReport1**, and click the **Add** button.
4. Install Export packages from **nuget** as follows:
   i) Go to **Tools** > **Nuget Package Manager** > **Manage Nuget Packages for Solution**...
   ii) Browse the following packages one by one and click Install.
   GrapeCity.ActiveReports.Export.Excel
   GrapeCity.ActiveReports.Export.Html
   GrapeCity.ActiveReports.Export.Image
   GrapeCity.ActiveReports.Export.Pdf
   GrapeCity.ActiveReports.Export.Word
   GrapeCity.ActiveReports.Export.Xml
5. Design your report.

To add code to the Web Form to export a report to PDF

1. Double-click on the design view of the aspx page. This creates an event-handling method for the Page_Load event.
2. Add code like the following to the Page_Load event.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the Page Load event.

```
Dim m_stream As New System.IO.MemoryStream()
Dim rpt As New GrapeCity.ActiveReports.SectionReport
Dim xtr As New System.Xml.XmlTextReader("\SectionReport1.rpx")
rpt.LoadLayout(xtr)
xtr.Close()
rpt.Run()
Dim PdfExport1 As New GrapeCity.ActiveReports.Export.Pdf.Section.PdfExport
PdfExport1.Export(rpt.Document, m_stream)
m_stream.Position = 0
Response.ContentType = "application/pdf"
Response.AddHeader("content-disposition", "attachment;filename=MyExport.pdf")
Response.BinaryWrite(m_stream.ToArray())
Response.End()
```

To write the code in C#

C# code. Paste INSIDE the Page Load event.

```
System.IO.MemoryStream m_stream = new System.IO.MemoryStream();
GrapeCity.ActiveReports.SectionReport rpt = new GrapeCity.ActiveReports.SectionReport();
System.Xml.XmlTextReader xtr = new System.Xml.XmlTextReader(Server.MapPath("") +
"\\SectionReport1.rpx");
```

```
rpt.LoadLayout(xtr);
xtr.Close();
rpt.Run();
GrapeCity.ActiveReports.Export.Pdf.Section.PdfExport pdfExport1 = new
GrapeCity.ActiveReports.Export.Pdf.Section.PdfExport();
pdfExport1.Export(rpt.Document, m_stream);
Response.ContentType = "application/pdf";
Response.AddHeader("content-disposition", "inline;filename=MyExport.pdf");
Response.BinaryWrite(m_stream.ToArray());
Response.End();
```

Note: To use the one-touch printing option, add the following to the code above.

Visual Basic.NET code. Paste INSIDE the Page Load event.

```
pdfExport1.Options.OnlyForPrint = True
```

C# code. Paste INSIDE the Page Load event.

```
pdfExport1.Options.OnlyForPrint = true;
```

To add code to the Web Form to export a report to Excel

1. Double-click on the design view of the aspx page. This creates an event-handling method for the Page_Load event.
2. Add code like the following to the Page_Load event.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the Page Load event.

```
Dim m_stream As New System.IO.MemoryStream()
Dim rpt As New GrapeCity.ActiveReports.SectionReport
Dim xtr As New System.Xml.XmlTextReader("\SectionReport1.rpx")
rpt.LoadLayout(xtr)
xtr.Close()
rpt.Run()
Dim XlsExport1 As New GrapeCity.ActiveReports.Export.Excel.Section.XlsExport
XlsExport1.MinColumnWidth = 0.5
XlsExport1.Export(rpt.Document, m_stream)
m_stream.Position = 0
Response.ContentType = "application/vnd.ms-excel"
Response.AddHeader("content-disposition", "inline; filename=MyExport.xls")
Response.BinaryWrite(m_stream.ToArray())
Response.End()
```

To write the code in C#

C# code. Paste INSIDE the Page Load event.

```
System.IO.MemoryStream m_stream = new System.IO.MemoryStream();
GrapeCity.ActiveReports.SectionReport rpt = new GrapeCity.ActiveReports.SectionReport();
```

```
System.Xml.XmlTextReader xtr = new System.Xml.XmlTextReader(Server.MapPath("") +
"\\SectionReport1.rpx");
rpt.LoadLayout(xtr);
xtr.Close();
rpt.Run();
GrapeCity.ActiveReports.Export.Excel.Section.XlsExport XlsExport1 = new
GrapeCity.ActiveReports.Export.Excel.Section.XlsExport();
XlsExport1.MinColumnWidth = 0.5f;
XlsExport1.Export(rpt.Document, m_stream);
m_stream.Position = 0;
Response.ContentType = "application/vnd.ms-excel";
Response.AddHeader("content-disposition","inline; filename=MyExport.xls");
Response.BinaryWrite(m_stream.ToArray());
Response.End();
```

To add code to the Web Form to export a report to TIFF

1. Double-click on the design view of the aspx page. This creates an event-handling method for the Page_Load event.
2. Add code like the following to the Page_Load event.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the Page Load event.

```
Dim m_stream As New System.IO.MemoryStream()
Dim rpt As New GrapeCity.ActiveReports.SectionReport
Dim xtr As New System.Xml.XmlTextReader(Server.MapPath("\SectionReport1.rpx"))
rpt.LoadLayout(xtr)
xtr.Close()
rpt.Run()
Dim TiffExport1 As New GrapeCity.ActiveReports.Export.Image.Tiff.Section.TiffExport
Me.TiffExport1.CompressionScheme =
GrapeCity.ActiveReports.Export.Image.Tiff.Section.CompressionScheme.None
Me.TiffExport1.Export(rpt.Document, m_stream)m_stream.Position = 0
Response.ContentType = "image/tiff"
Response.AddHeader("content-disposition", "inline; filename=MyExport.tiff")
Response.BinaryWrite(m_stream.ToArray())
Response.End()
```

To write the code in C#

C# code. Paste INSIDE the Page Load event.

```
System.IO.MemoryStream m_stream = new System.IO.MemoryStream();
GrapeCity.ActiveReports.SectionReport rpt = new GrapeCity.ActiveReports.SectionReport();
System.Xml.XmlTextReader xtr = new System.Xml.XmlTextReader(Server.MapPath("") +
"\\SectionReport1.rpx");
rpt.LoadLayout(xtr);
xtr.Close();
rpt.Run();
```

```
GrapeCity.ActiveReports.Export.Image.Tiff.Section.TiffExport tiffExport1 = new
GrapeCity.ActiveReports.Export.Image.Tiff.Section.TiffExport();
tiffExport1.CompressionScheme =
GrapeCity.ActiveReports.Export.Image.Tiff.Section.CompressionScheme.None;
tiffExport1.Export(rpt.Document, m_stream);
m_stream.Position = 0;
Response.ContentType = "image/tiff";
Response.AddHeader("content-disposition","inline; filename=MyExport.tiff");
Response.BinaryWrite(m_stream.ToArray());
Response.End();
```

To add code to the Web Form to export a report to RTF

1. Double-click on the design view of the aspx page. This creates an event-handling method for the Page_Load event.
2. Add code like the following to the Page_Load event.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the Page Load event.

```
Dim m_stream As New System.IO.MemoryStream()
Dim rpt As New GrapeCity.ActiveReports.SectionReport
Dim xtr As New System.Xml.XmlTextReader("\SectionReport1.rpx")
rpt.LoadLayout(xtr)
xtr.Close()
rpt.Run()
Dim RtfExport1 As New GrapeCity.ActiveReports.Export.Word.Section.RtfExport
RtfExport1.Export(rpt.Document, m_stream)
m_stream.Position = 0
Response.ContentType = "application/msword"
Response.AddHeader("content-disposition", "inline; filename=MyExport.rtf")
Response.BinaryWrite(m_stream.ToArray())
Response.End()
```

To write the code in C#

C# code. Paste INSIDE the Page Load event.

```
System.IO.MemoryStream m_stream = new System.IO.MemoryStream();
GrapeCity.ActiveReports.SectionReport rpt = new GrapeCity.ActiveReports.SectionReport();
System.Xml.XmlTextReader xtr = new System.Xml.XmlTextReader(Server.MapPath("") +
"\\SectionReport1.rpx");
rpt.LoadLayout(xtr);
xtr.Close();
rpt.Run();
GrapeCity.ActiveReports.Export.Word.Section.RtfExport rtfExport1  = new
GrapeCity.ActiveReports.Export.Word.Section.RtfExport();
rtfExport1.Export(rpt.Document, m_stream);
m_stream.Position = 0;
Response.ContentType = "application/msword";
```

```
Response.AddHeader("content-disposition","inline; filename=MyExport.rtf");
Response.BinaryWrite(m_stream.ToArray());
Response.End();
```

To add code to the Web Form to export a report to Plain Text

1. Double-click on the design view of the aspx page. This creates an event-handling method for the Page_Load event.
2. Add code like the following to the Page_Load event.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the Page Load event.

```
Dim m_stream As New System.IO.MemoryStream()
Dim rpt As New GrapeCity.ActiveReports.SectionReport
Dim xtr As New System.Xml.XmlTextReader("\SectionReport1.rpx")
rpt.LoadLayout(xtr)
xtr.Close()
rpt.Run()
Dim TextExport1 As New GrapeCity.ActiveReports.Export.Xml.Section.TextExport
TextExport1.Export(rpt.Document, m_stream)
m_stream.Position = 0
Response.ContentType = "text/plain"
Response.AddHeader("content-disposition", "attachment; filename=MyExport.txt")
Response.BinaryWrite(m_stream.ToArray())
Response.End()
```

To write the code in C#

C# code. Paste INSIDE the Page Load event.

```
System.IO.MemoryStream m_stream = new System.IO.MemoryStream();
GrapeCity.ActiveReports.SectionReport rpt = new GrapeCity.ActiveReports.SectionReport();
System.Xml.XmlTextReader xtr = new System.Xml.XmlTextReader(Server.MapPath("") +
"\\SectionReport1.rpx");
rpt.LoadLayout(xtr);
xtr.Close();
rpt.Run();
GrapeCity.ActiveReports.Export.Xml.Section.TextExport textExport1 = new
GrapeCity.ActiveReports.Export.Xml.Section.TextExport ();
textExport1.Export(rpt.Document, m_stream);
m_stream.Position = 0;
Response.ContentType = "text/plain";
Response.AddHeader("content-disposition", "attachment; filename=MyExport.txt");
Response.BinaryWrite(m_stream.ToArray());
Response.End();
```

To run the project

Press **F5** to run the project.

Custom HTML Outputter

You can create a custom HTML outputter for your ActiveReports ASP.NET Web Application.

> **Note:** You cannot create a custom HTML outputter for a page report because the html rendering extension does not support the custom output formatter.

This walkthrough is split up into the following activities:

- Creating a public class for the HTML outputter
- Adding code to create the Html Export object and export the report
- Adding a folder for report output

To create a public class for the HTML outputter

1. In the Solution Explorer window, right-click on your project name and select **Add**, then **New Item**.
2. In the **Add New Item** dialog that appears, select **Class**.
3. Change the name of the class to **MyCustomHtmlOutputter** and click the **Add** button.
4. This opens the code view of the class file where you can add the code needed to create the public class.
5. **For C# code**, add the IOutputHtml interface to MyCustomHtmlOutputter class.

C# code.

```
public class MyCustomHtmlOutputter: IOutputHtml
```

The following example shows what the complete code for the method looks like.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste JUST ABOVE the class.

```
Imports System
Imports System.IO
Imports System.Web
Imports System.Text
Imports GrapeCity.ActiveReports
Imports GrapeCity.ActiveReports.Export.Html
```

Visual Basic.NET code. Paste INSIDE the class.

```
Implements IOutputHtml
'The http context of the request.
Private context As System.Web.HttpContext = Nothing
'The directory in which to save filename--this ensures that the filename
'is unique.
Private dirToSave As System.IO.DirectoryInfo = Nothing
Public mainPage As String = ""
Public Sub New(ByVal context As System.Web.HttpContext)
If context Is Nothing Then
Throw New ArgumentNullException("context")
End If
Me.context = context
```

```
Dim dirName As String = context.Server.MapPath("ReportOutput")
Me.dirToSave = New DirectoryInfo(dirName)
End Sub
#Region "Implementation of IOutputHtml"
Public Function OutputHtmlData(ByVal info As HtmlOutputInfoArgs) As String Implements
IOutputHtml.OutputHtmlData
Dim temp As String = ""
Select Case info.OutputKind
Case HtmlOutputKind.BookmarksHtml
Case HtmlOutputKind.FramesetHtml
temp = Me.GenUniqueFileNameWithExtension(".html")
Dim fs As New FileStream(temp, FileMode.CreateNew)
Me.WriteStreamToStream(info.OutputStream, fs)
fs.Close()
Return temp
Case HtmlOutputKind.HtmlPage
'Store the name of the main page so we can redirect the
'browser to it
Me.mainPage = Me.GenUniqueFileNameWithExtension(".html")
Dim fs As New FileStream(Me.mainPage, FileMode.CreateNew)
Me.WriteStreamToStream(info.OutputStream, fs)
fs.Close()
Return Me.mainPage
Case HtmlOutputKind.ImageJpg
'Create a file with a .jpg extension:
temp = Me.GenUniqueFileNameWithExtension(".jpg")
Dim fs As New FileStream(temp, FileMode.CreateNew)
fs = File.Create(temp)
Me.WriteStreamToStream(info.OutputStream, fs)
fs.Close()
Return temp
Case HtmlOutputKind.ImagePng
'Create a file with a .png extension:
temp = Me.GenUniqueFileNameWithExtension(".png")
Dim fs As New FileStream(temp, FileMode.CreateNew)
Me.WriteStreamToStream(info.OutputStream, fs)
fs.Close()
Return temp
Case Else
'Default to html:
temp = Me.GenUniqueFileNameWithExtension(".html")
Dim fs As New FileStream(temp, FileMode.CreateNew)
Me.WriteStreamToStream(info.OutputStream, fs)
fs.Close()
Return temp
End Select
End Function Public Sub Finish() Implements IOutputHtml.Finish
End Sub
```

```
#End Region
Private Sub WriteStreamToStream(ByVal sourceStream As Stream, ByVal targetStream As
Stream)
'Find the size of the source stream:
Dim size As Integer = CType(sourceStream.Length, Integer)
'Create a buffer that same size
Dim buffer(size) As Byte
'Move the source stream to the beginning
sourceStream.Seek(0, SeekOrigin.Begin)
'Copy the sourceStream into our buffer
sourceStream.Read(buffer, 0, size)
'Write out the buffer to the target stream
targetStream.Write(buffer, 0, size)
End Sub
Private Function GenUniqueFileNameWithExtension(ByVal extensionWithDot As String) As
String
Dim r As New System.Random()
Dim unique As Boolean = False
Dim filePath As String = ""
Dim iRandom As Integer = 0
'Generate a random name until it's unique
While Not unique
iRandom = r.Next()
'Build the full filename
Dim sb = New StringBuilder()
sb.Append(Me.dirToSave.FullName)
sb.Append(Path.DirectorySeparatorChar)
sb.Append(iRandom.ToString())
sb.Append(extensionWithDot)
filePath = sb.ToString()
If File.Exists(filePath) = False Then
unique = True
Else
unique = False
End If
End While
Return filePath
End Function
End Class
```

To write the code in C#

C# code. Paste JUST ABOVE the class.

```
using System;
using System.IO;
using System.Web;
using System.Text;
```

```
using GrapeCity.ActiveReports;
using GrapeCity.ActiveReports.Export.Html;
```

C# code. Paste INSIDE the class.

```
   //The http context of the request
   private System.Web.HttpContext context = null;
    //The directory in which to save filename--this ensures that the filename
    //is unique.
   private System.IO.DirectoryInfo dirToSave = null;
   public string mainPage = "";
   public MyCustomHtmlOutputter(System.Web.HttpContext context)
   {
      if(context == null)
      {
        throw new ArgumentNullException("context");
          }
        this.context = context;
        string dirName = context.Server.MapPath("ReportOutput");
        this.dirToSave = new DirectoryInfo(dirName);
    }

#region Implementation of IOutputHtml
public string OutputHtmlData(HtmlOutputInfoArgs info)
{
   string temp = "";
   switch(info.OutputKind)
   {
      case HtmlOutputKind.BookmarksHtml:
      case HtmlOutputKind.FramesetHtml:
      {
        temp = this.GenUniqueFileNameWithExtension(".html");
        FileStream fs = File.Create(temp);
        this.WriteStreamToStream(info.OutputStream, fs);
        fs.Close();
        return temp;
      }

      case HtmlOutputKind.HtmlPage:
      {
         //Store the name of the main page so we can
         //redirect the browser to it
        this.mainPage = this.GenUniqueFileNameWithExtension(".html");
        FileStream fs = File.Create(this.mainPage);
        this.WriteStreamToStream(info.OutputStream, fs);
        fs.Close();
        return this.mainPage;
      }
```

```
      case HtmlOutputKind.ImageJpg:
      {
         // Create a file with a .jpg extension:
         temp = this.GenUniqueFileNameWithExtension(".jpg");
        FileStream fs = File.Create(temp);
        this.WriteStreamToStream(info.OutputStream, fs);
        fs.Close();
        return temp;
      }

      case HtmlOutputKind.ImagePng:
      {
        //Create a file with a .png extension:
        temp = this.GenUniqueFileNameWithExtension(".png");
        FileStream fs = File.Create(temp);
        this.WriteStreamToStream(info.OutputStream, fs);
        fs.Close();
        return temp;
      }

      default:
      {
        //Default to html:
        temp = this.GenUniqueFileNameWithExtension(".html");
        FileStream fs = File.Create(temp);
        this.WriteStreamToStream(info.OutputStream, fs);
        fs.Close();
        return temp;
      }
   }
}

public void Finish()
{
}
#endregion

private void WriteStreamToStream(Stream sourceStream, Stream targetStream)
{
   //Find the size of the source stream
   int size = (int)sourceStream.Length;

   //Create a buffer that same size
   byte[] buffer = new byte[size];

   //Move the source stream to the beginning
   sourceStream.Seek(0, SeekOrigin.Begin);
```

```
   //Copy the sourceStream into our buffer
   sourceStream.Read(buffer, 0, size);

   //Write out the buffer to the target stream
   targetStream.Write(buffer, 0, size);
}

private string GenUniqueFileNameWithExtension(string extensionWithDot)
{
    System.Random r = new Random();
    bool unique = false;
    string filePath = "";
    int iRandom = 0;
    //Generate a random name until it's unique
    while(!unique)
    {
       iRandom = r.Next();
       //Buld the full filename
       System.Text.StringBuilder sb = new System.Text.StringBuilder();
       sb.Append(this.dirToSave.FullName);
       sb.Append(Path.DirectorySeparatorChar);
       sb.Append(iRandom.ToString());
       sb.Append(extensionWithDot);
       filePath = sb.ToString();
       unique = !File.Exists(filePath);
     }
     return filePath;
  }
```

To add code to the Web Form to export to HTML

1. Add an Section Report (Code-based) to the project, and name it **rptCustHTML**.
2. Now add a Web form and double-click on the design view of the ASPX. This creates an event-handling method for the Web Form's Page Load event.
3. Add the following code to the Page Load event.

The following example shows what the code for the method looks like.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the Page Load event.

```
Dim rpt As New rptCustHTML()
Try

rpt.Run(False)
Catch eRunReport As Exception
'If the report fails to run, report the error to the user

Response.Clear()
```

```
Response.Write("<h1>Error running report:</h1>")
Response.Write(eRunReport.ToString())
Return
End Try
'Buffer this page's output until the report output is ready.

Response.Buffer = True
'Clear any part of this page that might have already been buffered for output.

Response.ClearContent()
'Clear any headers that might have already been buffered (such as the content type

'for an HTML page)

Response.ClearHeaders()
'Tell the browser and the "network" that the resulting data of this page should be

'cached since this could be a dynamic report that changes upon each request.

Response.Cache.SetCacheability(HttpCacheability.NoCache)
'Tell the browser this is an Html document so it will use an appropriate viewer.

Response.ContentType = "text/HTML"
'Create the Html export object

Dim HtmlExport1 As New GrapeCity.ActiveReports.Export.Html.Section.HtmlExport()
Dim outputter As New MyCustomHtmlOutputter(Me.Context)
HtmlExport1.Export(rpt.Document, outputter, "")
Response.Redirect("ReportOutput" + "/" + System.IO.Path.GetFileName(outputter.mainPage))
```

To write the code in C#

C# code. Paste INSIDE the Page Load event.

```
rptCustHTML rpt = new rptCustHTML();
try
{
   rpt.Run(false);
}
catch (Exception eRunReport)
{
   //If the report fails to run, report the error to the user
   Response.Clear();
   Response.Write("<h1>Error running report:</h1>");
   Response.Write(eRunReport.ToString());
   return;
}
//Buffer this page's output until the report output is ready.
```

```
    Response.Buffer = true;

    //Clear any part of this page that might have already been buffered for output.
    Response.ClearContent();

    //Clear any headers that might have already been buffered (such as the content
    //type for an HTML page)
    Response.ClearHeaders();

    //Tell the browser and the "network" that the resulting data of this page should
    //be cached since this could be a dynamic report that changes upon each request.
    Response.Cache.SetCacheability(HttpCacheability.NoCache);

    //Tell the browser this is an Html document so it will use an appropriate viewer.
    Response.ContentType = "text/html";

    //Create the HTML export object
    GrapeCity.ActiveReports.Export.Html.Section.HtmlExport htmlExport1 = new
GrapeCity.ActiveReports.Export.Html.Section.HtmlExport();


    //Export the report to HTML in this session's webcache
    MyCustomHtmlOutputter outputter = new MyCustomHtmlOutputter(this.Context);
    htmlExport1.Export(rpt.Document, outputter, "");
    Response.Redirect("ReportOutput" + "/" +
System.IO.Path.GetFileName(outputter.mainPage));
```

To add a folder to the project for report output

1. In the Solution Explorer, right-click your solution and select **Add**, then **New Folder**.
2. Name the folder **ReportOutput**.
3. Ensure that you have write permissions for this folder.
4. To view the results in your Web browser, run the project.

Script

This section contains the following walkthroughs that fall under the Script category.

ActiveReports allows you to embed script in reports so that code becomes portable when you save a report layout to XML-based RPX format. This characteristic allows the options of stand-alone reporting and web reporting without the need to distribute related .vb or .cs files.

By embedding script when the report is saved as an RPX file, it can later by loaded, run and displayed directly to the viewer control without using the designer. Script can also be used in conjunction with RPX files to allow distributed reports to be updated without recompiling the Visual Studio project.

Script for Simple Reports
: This walkthrough demonstrates how to embed script in a simple stand-alone report.

Script for Subreports

This walkthrough demonstrates how to embed script to pass a parameter to a subreport.

Script for Simple Reports

ActiveReports allows you to use scripting to embed code in reports saved to the XML-based RPX file format. By embedding script in reports saved as RPX files, you can later load, run, and display reports directly in the viewer control without using the designer. This walkthrough illustrates how to include scripting in a simple report.

This walkthrough is split into the following activities:

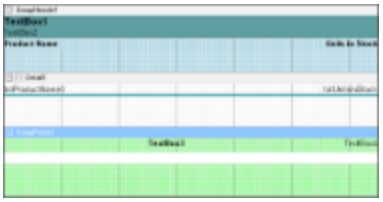
- Temporarily connecting the report to a data source
- Adding controls to a report to display data
- Adding scripting to supply data for the controls
- Saving the report to an RPX file

Tip: For basic steps like adding a report to a Visual Studio project and viewing a report, please see the Basic Data Bound Reports walkthrough.

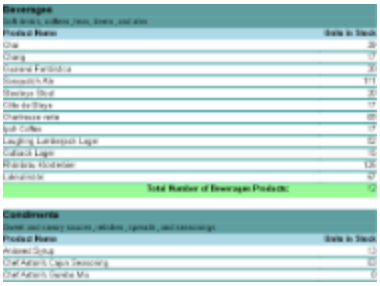
Note: This walkthrough uses the the Northwind database. The NWIND.mdb file can be downloaded from GitHub: ..\Samples14\Data\NWIND.mdb.

When you have finished this walkthrough, you will have a report that looks similar to the following at design time and at run time.

Design-Time Layout



Run-Time Layout



To add an ActiveReport to the Visual Studio project

1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 Section Report (code-based)** and in the Name field, rename the file as **rptSimpleScript**.

4. Click the **Add** button to open a new section report in the designer.

See Quick Start for information on adding different report layouts.

To connect the report to a data source

> **Note**: The following steps are just for convenience so that the fields list in the Report Explorer can be populated at design time.

1. In the Report Data Source dialog, on the **OLE DB** tab, next to Connection String, click the Build button.
2. In the Data Link Properties window that appears, select **Microsoft Jet 4.0 OLE DB Provider** and click the **Next** button to move to the Connection tab.
3. Click the ellipsis (...) button to browse to your database, for example the NWind.mdb sample database. Click **Open** once you have selected the appropriate database path.
4. Click the **Test Connection** button to see if you have successfully connected to the database.
5. Click **OK** to close the Data Link Properties window and return to the Report Data Source dialog. Notice that the Connection String field gets filled automatically.
6. In the **Query** field on the **OLE DB** tab, enter the following SQL query.

SQL Query
`SELECT * FROM categories INNER JOIN products ON categories.categoryid = products.categoryid ORDER BY products.categoryid, products.productid`

7. Click **OK** to save the data source and return to the report design surface.

To create a layout for the report

1. Right-click the design surface of the report and select **Insert** then **Group Header/Footer** to add group header and footer sections to your report.
2. Increase the group header section's height so that you have room to work.
3. With the GroupHeader section selected, go to the Properties Window to set the following properties.

Property Name	Property Value
BackColor	LightBlue
CanShrink	True
DataField	CategoryName
GroupKeepTogether	All
KeepTogether	True

4. From the toolbox, drag the following controls to the GroupHeader section and set the properties of each control as indicated.

   TextBox1

Property Name	Property Value
DataField	CategoryName
Location	0, 0 in
Size	6.5, 0.2 in

BackColor	CadetBlue
Font	Bold:True
Font Size	12

TextBox2

Property Name	Property Value
DataField	Description
Location	0, 0.2 in
Size	6.5, 0.2 in
BackColor	CadetBlue

Label1

Property Name	Property Value
Text	Product Name
Location	0, 0.4 in
Size	1, 0.2 in
Font	Bold:True

Label2

Property Name	Property Value
Text	Units in Stock
Location	5.5, 0.4 in
Size	1, 0.2 in
Font	Bold:True
Alignment	Right

5. From the toolbox, drag the following controls onto the detail section and set the properties of each as indicated.

TextBox1

Property Name	Property Value
DataField	ProductName
Location	0, 0 in
Size	5.5, 0.2 in

TextBox2

Property Name	Property Value
DataField	UnitsInStock
Location	5.5, 0 in
Size	1, 0.2 in
Alignment	Right

6. Click just below the fields to select the Detail section, and in the Properties Window, set the **CanShrink** property to **True** to eliminate white space in the rendered report.
7. In the Detail section, select both TextBox1 and TextBox2, right-click and select **Format Border**.
    - Select DarkCyan in the color combo box.
    - Select the solid line in the Line Styles pane.
    - Click the bottom edge in the Preview pane.
    - Click the **OK** button to add a solid cyan line to the bottom edge of the text boxes.
8. Increase the group footer section's height so that you have room to work.
9. With the GroupFooter section selected, go to the properties window and set the following properties.

Property Name	Property Value
BackColor	PaleGreen
CanShrink	True

10. From the toolbox, drag the following controls to the GroupFooter Section and set the properties of each control as indicated.

TextBox1

Property Name	Property Value
DataField	TotalLabel
Location	2.5, 0 in
Size	3, 0.2 in
Font	Bold:True

TextBox2

Property Name	Property Value
DataField	ProductName
Location	5.5, 0 in
SummaryType	Subtotal
SummaryFunc	Count
SummaryRunning	Group
SummaryGroup	GroupHeader1
Alignment	Right

Label1

Property Name	Property Value
Location	0, 0.25 in
Size	6.5, 0.2 in
BackColor	White (creates white space after the subtotal)
Text	**Note**: Delete the default text.

To add scripting to the report to supply data for the controls

1. Click in the grey area below the report to select it, and in the Properties Window, change the **ScriptLanguage** property for the report to the scripting language you want to use. The default setting is **C#**.
2. Click the **Script** tab located at the bottom edge of the report designer to access the scripting editor. Add the scripting code.

The following example shows what the scripting code looks like.

Warning: Do not access the Fields collection outside the **DataInitialize** and **FetchData** events. Accessing the Fields collection outside of these events is not supported, and has unpredictable results.

To write the script in Visual Basic.NET.

Visual Basic.NET script. Paste in the script editor window.

```
Private Shared m_reader As System.Data.OleDb.OleDbDataReader
Private Shared m_cnn As System.Data.OleDb.OleDbConnection

Public Sub ActiveReport_ReportStart()
  'Set up a data connection for the report
  rpt.DataSource = ""
  Dim connString As String = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=[User
Folder]\Samples14\Data\NWIND.mdb"
  Dim sqlString As String = "SELECT * FROM categories INNER JOIN products ON
categories.categoryid = products.categoryid ORDER BY products.categoryid,
products.productid"

  m_cnn = new System.Data.OleDb.OleDbConnection(connString)
  Dim m_Cmd As System.Data.OleDb.OleDbCommand = new
System.Data.OleDb.OleDbCommand(sqlString, m_cnn)

  If m_cnn.State = System.Data.ConnectionState.Closed Then
     m_cnn.Open
  End If
  m_reader = m_Cmd.ExecuteReader
End Sub
```

```
Public Sub ActiveReport_DataInitialize()
  'Add data fields to the report
  rpt.Fields.Add("CategoryID")
  rpt.Fields.Add("CategoryName")
  rpt.Fields.Add("ProductName")
  rpt.Fields.Add("UnitsInStock")
  rpt.Fields.Add("Description")
  rpt.Fields.Add("TotalLabel")
End Sub

Public Function ActiveReport_FetchData(ByVal eof As Boolean) As Boolean
  Try
    m_reader.Read
    'Populated the fields with data from the data reader
    rpt.Fields("CategoryID").Value = m_reader("categories.CategoryID")
    rpt.Fields("CategoryName").Value = m_reader("CategoryName")
    rpt.Fields("ProductName").Value = m_reader("ProductName")
    rpt.Fields("UnitsInStock").Value = m_reader("UnitsInStock")
    rpt.Fields("Description").Value = m_reader("Description")
    'Concatenate static text with data
    rpt.Fields("TotalLabel").Value = "Total Number of " + m_reader("CategoryName")+ "
Products:"
    eof = False
  Catch
    'If the end of the data file has been reached, tell the FetchData function
    eof = True
  End Try
 Return eof
End Function

Public Sub ActiveReport_ReportEnd()
  'Close the data reader and connection
  m_reader.Close
  m_cnn.Close
End Sub
```

To write the script in C#.

C# script. Paste in the script editor window.

```
//C#
private static System.Data.OleDb.OleDbDataReader m_reader;
private static System.Data.OleDb.OleDbConnection m_cnn;

public void ActiveReport_ReportStart()
{
   //Set up a data connection for the report
```

```
   rpt.DataSource = "";
   string m_cnnString = @"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Users\[User
Folder]\Samples14\Data\NWIND.mdb";
   string sqlString = "SELECT * FROM categories INNER JOIN products ON
categories.categoryid = products.categoryid ORDER BY products.categoryid,
products.productid";
   m_cnn = new System.Data.OleDb.OleDbConnection(m_cnnString);
   System.Data.OleDb.OleDbCommand m_Cmd = new
System.Data.OleDb.OleDbCommand(sqlString,m_cnn);

   if(m_cnn.State == System.Data.ConnectionState.Closed)
   {
      m_cnn.Open();
   }
   m_reader = m_Cmd.ExecuteReader();
}

public void ActiveReport_DataInitialize()
{
   //Add data fields to the report
   rpt.Fields.Add("CategoryID");
   rpt.Fields.Add("CategoryName");
   rpt.Fields.Add("ProductName");
   rpt.Fields.Add("UnitsInStock");
   rpt.Fields.Add("Description");
   rpt.Fields.Add("TotalLabel");
}

public bool ActiveReport_FetchData(bool eof)
{
   try
   {
     m_reader.Read();
     //Populated the fields with data from the data reader
     rpt.Fields["CategoryID"].Value = m_reader["categories.CategoryID"].ToString();
     rpt.Fields["CategoryName"].Value = m_reader["CategoryName"].ToString();
     rpt.Fields["ProductName"].Value = m_reader["ProductName"].ToString();
     rpt.Fields["UnitsInStock"].Value = m_reader["UnitsInStock"].ToString();
     rpt.Fields["Description"].Value = m_reader["Description"].ToString();
     //Concatenate static text with data
     rpt.Fields["TotalLabel"].Value = "Total Number of " +
m_reader["CategoryName"].ToString() + " Products:";
     eof = false;
   }
   catch
   {
     //If the end of the data file has been reached, tell the FetchData function
     eof = true;
```

```
    }
    return eof;
}

public void ActiveReport_ReportEnd()
{
    //Close the data reader and connection
    m_reader.Close();
    m_cnn.Close();
}
```

To save the report to an XML-based RPX file

1. From the **Report** menu, select **Save Layout**.
2. In the Save dialog that appears, enter a name for the report, i.e. **rptScript.rpx**, and click the **Save** button.

To view the report

- Click the preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

Script for Subreports

ActiveReports allows you to use scripting to permit reports saved to an XML file to contain code. By including scripting when reports are saved into XML, the reports later can be loaded, run, and displayed directly to the viewer control without needing to use the designer.

This walkthrough illustrates how to use scripting when creating a subreport.

This walkthrough is split up into the following activities:

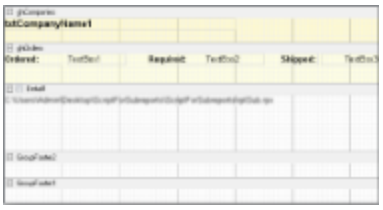
- Temporarily connecting the main report to a data source
- Connecting the subreport to a data source
- Adding controls to each report to display data
- Adding the scripting code for rptMain
- Viewing the report

> **Tip**: For basic steps like adding a report to a Visual Studio project and viewing a report, please see the Basic Data Bound Reports walkthrough.

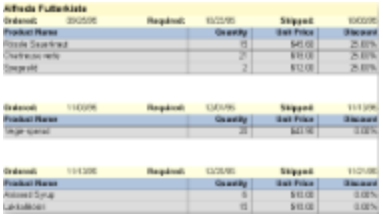
> **Note:** This walkthrough uses the Northwind database. The NWIND.mdb file can be downloaded from GitHub: ..\Samples14\Data\NWIND.mdb.

When you have finished this walkthrough, you will have a report that looks similar to the following at design time and at run time.

Design-Time Layout (main report)



Run-Time Layout (main report)



To add an ActiveReport to the Visual Studio project

1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 Section Report (xml-based)** and in the Name field, rename the file as **rptMain**.
4. Click the **Add** button to open a new section report in the designer.

See Quick Start for information on adding different report layouts.

To connect the report to a data source

> **Note**: The following steps are just for convenience so that the fields list in the Report Explorer can be populated at design time.

1. In the Report Data Source dialog, on the **OLE DB** tab, next to Connection String, click the Build button.
2. In the Data Link Properties window that appears, select **Microsoft Jet 4.0 OLE DB Provider** and click the **Next** button to move to the Connection tab.
3. Click the ellipsis (...) button to browse to your database, for example the NWind.mdb sample database. Click **Open** once you have selected the appropriate database path.
4. Click the **Test Connection** button to see if you have successfully connected to the database.
5. Click **OK** to close the Data Link Properties window and return to the Report Data Source dialog. Notice that the Connection String field gets filled automatically.
6. In the **Query** field on the **OLE DB** tab, enter the following SQL query.

SQL Query
`SELECT * FROM Orders INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID ORDER BY CompanyName, OrderDate`

7. Click **OK** to save the data source and return to the report design surface.

To add a report for the subreport

1. From the **Project** menu, select **Add New Item**.
2. In the Add New Item dialog that appears, select **ActiveReports 14 Section Report (xml-based)** and in the Name field, rename the file as **rptSub**.

3. Click the **Add** button to open a new section report in the designer.
4. Right-click the PageHeader or PageFooter section and select **Delete**. Subreports do not render these sections, so deleting them saves processing time.
5. Click in the grey area below the report to select it, and in the Properties window, change the report's **ShowParameterUI** property to **False**. This prevents the subreport from requesting a parameter from the user.

See Quick Start for information on adding different report layouts.

To connect the subreport to a data source

> **Note**: The following steps are just for convenience so that the fields list in the Report Explorer can be populated at design time.

1. In the Report Data Source dialog, on the **OLE DB** tab, next to Connection String, click the Build button.
2. In the Data Link Properties window that appears, select **Microsoft Jet 4.0 OLE DB Provider** and click the **Next** button to move to the Connection tab.
3. Click the ellipsis (...) button to browse to your database, for example the NWind.mdb sample database. Click **Open** once you have selected the appropriate database path.
4. Click the **Test Connection** button to see if you have successfully connected to the database.
5. Click **OK** to close the Data Link Properties window and return to the Report Data Source dialog. Notice that the Connection String field gets filled automatically.
6. In the **Query** field on the **OLE DB** tab, enter the following SQL query.

SQL Query
`SELECT * FROM [order details] inner join products on [order details].productid = products.productid`

7. Click **OK** to save the data source and return to the report design surface.

To create a layout for the main report

1. Right-click the design surface of rptMain and select **Insert** then **Group Header/Footer** to add group header and footer sections to the report.
2. In the Properties Window, make the following changes to the group header.

Property Name	Property Value
Name	ghCompanies
BackColor	LemonChiffon
CanShrink	True
DataField	CompanyName
GroupKeepTogether	All
KeepTogether	True

3. In the Report Explorer, expand the **Fields** node, then the **Bound** node. Drag the **CompanyName** field onto ghCompanies and in the Properties window, set the properties as follows.

Property Name	Property Value
Size	4, 0.2 in
Location	0, 0 in

Font Bold	True
Font Size	12

4. Right-click the design surface of rptMain and select **Insert** then **Group Header/Footer** to add the second group header and footer sections to the report.
5. In the Properties Window, make the following changes to the second group header.

Property Name	Property Value
Name	ghOrders
BackColor	LightYellow
CanShrink	True
DataField	OrderDate
GroupKeepTogether	All
KeepTogether	True

6. From the toolbox, drag three TextBox controls onto **ghOrders** and set the properties for each control as follows.

TextBox1

Property Name	Property Value
DataField	OrderDate
Location	1.1, 0 in
Size	1, 0.2 in
OutputFormat	MM/dd/yy

TextBox2

Property Name	Property Value
DataField	RequiredDate
Location	3.5, 0 in
Size	1, 0.2 in
OutputFormat	MM/dd/yy

TextBox3

Property Name	Property Value
DataField	ShippedDate
Location	5.5, 0 in
Size	1, 0.2 in
OutputFormat	MM/dd/yy
Alignment	Right

7. From the toolbox, drag three Label controls onto **ghOrders** and set the properties for each control as follows.

Label1

Property Name	Property Value
Location	0, 0 in
Size	1, 0.2 in
Text	Ordered:
Font	Bold:True

Label2

Property Name	Property Value
Location	2.5, 0 in
Size	1, 0.2 in
Text	Required:
Font	Bold:True

Label3

Property Name	Property Value
Location	4.8, 0 in
Size	0.65, 0.2 in
Text	Shipped:
Font	Bold:True

8. Select the Detail section and in the Properties window, set the **CanShrink** property to **True**.
9. From the toolbox, drag the Subreport control onto the Detail section and in the Properties window, set the properties as follows.

Property Name	Property Value
ReportName	***full project path***\rptSub.rpx
Name	SubReport1
Size	6.5, 1 in
Location	0, 0 in

To create a layout for the subreport

1. Right-click the design surface of rptSub and select **Insert** then **Group Header/Footer** to add group header and footer sections to the report.
2. In the Properties window, make the following changes to the group header.

Property Name	Property Value

Name	ghOrderDetails
BackColor	LightSteelBlue
CanShrink	True
DataField	OrderID

3. From the toolbox, drag four label controls to **ghOrderDetails** and set the properties for each label as follows.

Label1

Property Name	Property Value
Location	0, 0 in
Text	Product Name
Font	Bold:True
Alignment	Left

Label2

Property Name	Property Value
Location	3.25, 0 in
Text	Quantity
Font	Bold:True
Alignment	Right

Label3

Property Name	Property Value
Location	4.4, 0 in
Text	Unit Price
Font	Bold:True
Alignment	Right

Label4

Property Name	Property Value
Location	5.5, 0 in
Text	Discount
Font	Bold:True
Alignment	Right

4. From the toolbox, drag four Line controls to **ghOrderDetails** and set the properties for each line as follows.

Line1

Property Name	Property Value
X1	3.2
X2	3.2
Y1	0
Y2	0.2

Line2

Property Name	Property Value
X1	4.3
X2	4.3
Y1	0
Y2	0.2

Line3

Property Name	Property Value
X1	5.45
X2	5.45
Y1	0
Y2	0.2

Line4

Property Name	Property Value
X1	0
X2	6.5
Y1	0.2
Y2	0.2

5. Click the Detail section and in the Properties window, set the following properties.

Property Name	Property Value
BackColor	Gainsboro
CanShrink	True

6. From the toolbox, drag four TextBox controls onto onto the Detail section and set the properties as follows.

TextBox1

Property Name	Property Value
DataField	ProductName
Location	0, 0 in
Size	3.15, 0.2 in
Alignment	Left

TextBox2

Property Name	Property Value
DataField	Quantity
Location	3.25, 0 in
Size	1, 0.2 in
Alignment	Right

TextBox3

Property Name	Property Value
DataField	Products.UnitPrice
Location	4.4, 0 in
Size	1, 0.2 in
Alignment	Right
OutputFormat	Currency

TextBox4

Property Name	Property Value
DataField	Discount
Location	5.5, 0 in
Size	1, 0.2 in
Alignment	Right
OutputFormat	Percentage

7. From the toolbox, drag four Line controls to the Detail section and set the properties as follows.

Line5

Property Name	Property Value
X1	3.2
X2	3.2

Y1	0
Y2	0.2

Line6

Property Name	Property Value
X1	4.3
X2	4.3
Y1	0
Y2	0.2

Line7

Property Name	Property Value
X1	5.45
X2	5.45
Y1	0
Y2	0.2

Line8

Property Name	Property Value
X1	0
X2	6.5
Y1	0.2
Y2	0.2

To embed script in the main report

1. Change the **ScriptLanguage** property for the report to the appropriate scripting language. The default setting is C#.
2. Click the Script tab located below the report designer to access the scripting editor.
3. Embed script to set the data source for the main report and pass data into the subreport.

The following example shows what the script looks like.

To write the script in Visual Basic.NET

Visual Basic.NET script. Paste in the script editor window.

```
Dim rptSub As GrapeCity.ActiveReports.SectionReport
Sub ActiveReport_ReportStart
  'Create a new instance of the generic report
  rptSub = new GrapeCity.ActiveReports.SectionReport()
```

```
  'Load the rpx file into the generic report
  rptSub.LoadLayout(me.SubReport1.ReportName)
  'Connect data to the main report
  Dim connString As String = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=[User
Folder]\Samples14\Data\NWIND.mdb;Persist Security Info=False"
  Dim sqlString As String = "Select * from orders inner join customers on
orders.customerid = customers.customerid order by CompanyName,OrderDate"
  Dim ds As new GrapeCity.ActiveReports.Data.OleDBDataSource()
  ds.ConnectionString = connString
  ds.SQL = sqlString
  rpt.DataSource = ds
End Sub

Sub Detail_Format
  Dim rptSubCtl As GrapeCity.ActiveReports.SubReport = me.SubReport1
  Dim childDataSource As New GrapeCity.ActiveReports.Data.OleDBDataSource()
  childDataSource.ConnectionString = CType(rpt.DataSource,
GrapeCity.ActiveReports.Data.OleDBDataSource).ConnectionString
  'Set a parameter in the SQL query
  childDataSource.SQL = "Select * from [order details] inner join products on [order
details].productid = products.productid where [order details].orderid = <%OrderID%>"
  'Pass the data to the subreport
  rptSub.DataSource = childDataSource
  'Display rptSub in the subreport control
  rptSubCtl.Report = rptSub
End Sub
```

To write the script in C#

C# code. Paste in the script editor window.

```
GrapeCity.ActiveReports.SectionReport rptSub;
public void Detail_Format()
{
  GrapeCity.ActiveReports.SectionReportModel.SubReport rptSubCtl = this.SubReport1;
  GrapeCity.ActiveReports.Data.OleDBDataSource childDataSource = new
GrapeCity.ActiveReports.Data.OleDBDataSource();
  childDataSource.ConnectionString = ((GrapeCity.ActiveReports.Data.OleDBDataSource)
rpt.DataSource).ConnectionString;
  //Set a parameter in the SQL query
  childDataSource.SQL = "Select * from [order details] inner join products on [order
details].productid = products.productid where [order details].orderid = <%OrderID%>";
  //Pass the data to the subreport
  rptSub.DataSource = childDataSource;
  //Display rptSub in the subreport control
  rptSubCtl.Report = rptSub;
}
```

```
public void ActiveReport_ReportStart()
{
  //Create a new instance of the generic report
  rptSub = new GrapeCity.ActiveReports.SectionReport();
  //Load the rpx file into the generic report
  rptSub.LoadLayout(this.SubReport1.ReportName);
  //Connect data to the main report
  string connString = @"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=[User
Folder]\Samples14\Data\NWIND.mdb;Persist Security Info=False";
  string sqlString = "Select * from orders inner join customers on orders.customerid =
customers.customerid order by CompanyName,OrderDate";
  GrapeCity.ActiveReports.Data.OleDBDataSource ds = new
GrapeCity.ActiveReports.Data.OleDBDataSource();
  ds.ConnectionString = connString;
  ds.SQL = sqlString;
  rpt.DataSource = ds;
}
```

To view the report

- Click the preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information on how to load the xml-based section report onto the viewer.

Parameters

This section contains the following walkthroughs that fall under the Parameter category.

Using Parameters in Sub Reports
: This walkthrough demonstrates how to link a report with a SubReport using parameters.

Parameters for Charts
: This walkthrough demonstrates how to link a report with a chart using parameters.

Using Parameters in Sub Reports

Using parameters in SubReport, you can connect a SubReport to the parent report. By setting the parameter on the field that binds the parent report to SubReport, the parent report passes the data to display in SubReport through a parameter. This walkthrough illustrates the method to link the main report with a SubReport using parameters.

This walkthrough is split up into the following activities:

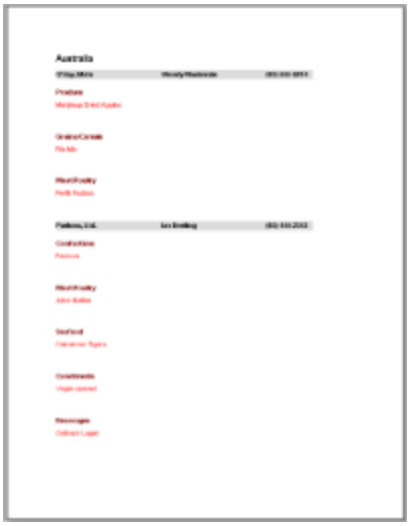
- Adding a parent report and a SubReport to a Visual Studio project
- Connecting the parent report to a data source
- Connecting the SubReport to a data source using parameter
- Adding controls to create layout of the parent report (rptParent)

- Adding controls to create a layout of the child report (rptChild)
- Adding code to embed the SubReport to the SubReport control in parent report
- Adding code to set the ShowParametersUI property of SubReport to False.
- Viewing the report

Note: This walkthrough uses tables from the NorthWind database. The NWIND.mdb file can be downloaded from GitHub: ..\Samples14\Data\NWIND.mdb.

Caution: SubReports do not render PageHeader and PageFooter sections.

When you complete this walkthrough you get a layout that looks similar to the following:



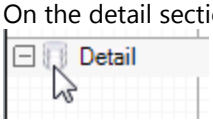
To add an ActiveReport to the Visual Studio project

1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 Section Report (code-based)** and in the Name field, rename the file as **rptParent**.
4. Click the **Add** button to open a new section report in the designer.
5. From the **Project** menu, select **Add New Item**.
6. In the Add New Item dialog that appears, select **ActiveReports 14 Section Report (code-based)** and in the Name field, rename the file as **rptChild**.
7. Click the **Add** button to open a second new section report in the designer.

See Quick Start for information on adding different report layouts.

To connect the parent report (rptParent) to a data source

1. On the detail section band, click the Data Source Icon.



2. In the Report Data Source dialog that appears, from the **OLE DB** tab, create a data source connection. See Bind Reports to a Data Source for further details.
3. Once the connection string field is populated, in the Query field, enter the following SQL query.

SQL Query
`Select * from suppliers order by country`

4. Click **OK** to save the data source and return to the report design surface.

To connect the child report (rptChild) to a data source using parameter

1. On the detail section band, click the Data Source Icon.

2. In the Report Data Source dialog that appears, from the **OLE DB** tab, create a data source connection. See Bind Reports to a Data Source for further details.
3. Once the connection string field is populated, in the Query field, enter the following parameterized SQL query.

SQL Query
`SELECT * FROM products INNER JOIN categories ON products.categoryid = categories.categoryid WHERE Products.SupplierID = <%SupplierID%>`

4. Click **OK** to save the data source and return to the report design surface.

To create a layout for the parent report (rptParent)

1. On the design surface, right click the PageHeader or PageFooter section and select **Delete** to remove the PageHeader/Footer pair.
2. Right click on the design surface of the parent report and insert **GroupHeader/Footer** section pair.
3. Click the GroupHeader section to select it and go to the Properties window to set the following properties.

Property Name	Property Value
Name	ghSuppliers
DataField	Country

4. From the toolbox, drag a TextBox control onto the groupHeader section and in the Properties window, set the properties as follows:

Property Name	Property Value
DataField	Country
Name	txtCountry
Text	Country
Location	0, 0
Font	Name:Arial, Size:13pt, Bold:True

5. Click the Detail section to select it and go to the Properties window to set the **CanShrink** property to **True**.
6. From the Visual Studio toolbox, drag and drop the following controls onto the detail section of the report and in the Properties window, set their properties as given below:

TextBox1

Property Name	Property Value
DataField	CompanyName

Name	txtCompanyName
Text	Company Name
Location	0.0625, 0.0625
Size	2.25, 0.2 in
BackColor	Silver
Font	Bold:True

TextBox2

Property Name	Property Value
DataField	ContactName
Name	txtContactName
Text	Contact Name
Location	2.312, 0.0625
Size	1.708, 0.2 in
BackColor	Silver
Font	Bold:True

TextBox3

Property Name	Property Value
DataField	Phone
Name	txtPhone
Text	Phone
Location	4.562, 0.0625
Size	1.542, 0.2 in
BackColor	Silver
Font	Bold:True

SubReport

Property Name	Property Value
Name	Subreport1
ReportName	ProductName
Location	0.0625, 0.312

To create a layout for the child report (rptChild)

1. On the design surface, right click the PageHeader or PageFooter section and select **Delete** to remove the PageHeader/Footer pair.
2. Right click on the design surface of the child report and insert **GroupHeader/Footer** section pair.
3. Click the GroupHeader section to select it and go to the Properties window to set the following properties.

Property Name	Property Value
Name	ghProducts
DataField	CategoryName

4. From the toolbox, drag and drop a TextBox control onto the groupHeader section and in the Properties window, set the following properties.

Property Name	Property Value
DataField	CategoryName
Name	txtCategoryName
Text	Category Name
Location	0.0625, 0.0625
Size	2.042, 0.2 in
ForeColor	Maroon
Font	Bold:True

5. Click the Detail section to select it and go to the Properties window to set the **CanShrink** property to **True**.
6. From the toolbox, drag and drop a TextBox control onto the detail section and in the Properties window, set the following properties.

Property Name	Property Value
DataField	ProductName
Name	txtProductName
Text	Product Name
Location	0.0625, 0.0625
Size	1.99, 0.2 in
ForeColor	Red

To connect the child report (rptChild) to the SubReport control in parent report (rptParent)

1. Double-click the gray area around the parent report (rptParent) to create an event-handling method for the **ReportStart** event.
2. Add code like the following to the handler to create a new instance of the child report (rptChild).

   To write the code in Visual Basic

Visual Basic.NET code. Paste JUST ABOVE the ReportStart event.
`Dim rpt As rptChild`
Visual Basic.NET code. Paste INSIDE the ReportStart event.

```
rpt = New rptChild()
```

To write the code in C#

C# code. Paste JUST ABOVE the ReportStart event.

```
private rptChild rpt;
```

C# code. Paste INSIDE the ReportStart event.

```
rpt = new rptChild();
```

3. Double-click the detail section of the parent report (rptParent) to create a detail_Format event.
4. Add code like the following to the handler to display a report in the SubReport control.

To write the code in Visual Basic

Visual Basic.NET code. Paste INSIDE the Format event.

```
Me.SubReport1.Report = rpt
```

To write the code in C#

C# code. Paste INSIDE the Format event.

```
this.subReport1.Report = rpt;
```

To set the ShowParametersUI property of SubReport to False

1. Click the gray area around the child report (rptChild) and go to the Properties window to set the **ShowParameterUI** property to **False**.

To view the report

- Click the preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

Parameters for Charts

Using parameters you can connect your report to the chart control. By setting the parameter on the field that connects the report to the chart, the report passes the data to display in the chart. This walkthrough illustrates how to link report with chart using parameters.

The walkthrough is split up into the following activities:

- Adding an ActiveReport to a Visual Studio project
- Connecting the report to a data source
- Adding controls to the report to display data

- Connecting the Chart control to a data source using parameter
- Setting the Chart's properties
- Viewing the report.

When you complete this walkthrough you get a layout that looks similar to the following at run time:

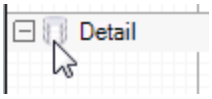
Note: This topic uses table from the NorthWind database. The NWIND.mdb file can be downloaded from GitHub: ..\Samples14\Data\NWIND.mdb.

To add an ActiveReport to the Visual Studio project

1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 14 Section Report (code-based)** and in the Name field, rename the file as **rptChartParams**.
4. Click the **Add** button to open a new section report in the designer.

See Quick Start for information on adding different report layouts.

To connect the report (rptChartParams) to a data source

1. On the detail section band, click the Data Source Icon.
   

2. In the Report Data Source dialog that appears, from the **OLE DB** tab, create a data source connection. See Bind Reports to a Data Source for further details.
3. Once the connection string field is populated, in the Query field, enter the following SQL query.

SQL Query
`SELECT * FROM Products ORDER BY CategoryID, ProductName`

4. Click **OK** to save the data source and return to the report design surface.

To add controls to the report to display data

1. On the design surface, right click the PageHeader or PageFooter section and select **Delete** to remove the

PageHeader/Footer pair.
2. Right click on the design surface and insert GroupHeader/Footer section pair.
3. Click the GroupHeader section to select it and go to the Properties window to set the following properties:

Property Name	Property Value
Name	ghCategoryID
DataField	CategoryID
GroupKeepTogether	All
Height	5.65 inches

4. Set Height property of Detail section to 0.23 inches and of GroupFooter section to 0 inch.
5. From the Visual Studio toolbox, drag and drop the following controls onto the GroupHeader section and set their properties as given below:

Label1

Property Name	Property Value
Name	lblCategoryID
Text	CategoryID
Location	1.78, 0

Label2

Property Name	Property Value
Name	lblProductName
Text	Product Name
Location	0.23, 5.43

Label3

Property Name	Property Value
Name	lblUnitsInStock
Text	Inventory stock
Location	5, 5.43

TextBox

Property Name	Property Value
DataField	CategoryID
Name	txtCategoryID
Text	CategoryID
Location	3.72, 0

Chart

Property Name	Property Value
Name	ChartControl
Location	0, 0.313
Size	6.5, 4.66

6. From the Visual Studio toolbox, drag and drop the following controls onto the detail section and set their properties as given below:

TextBox1

Property Name	Property Value
DataField	ProductName
Name	txtProductName
Text	ProductName
Location	0.23, 0

TextBox2

Property Name	Property Value
DataField	UnitsInStock
Name	txtUnitsInStock
Text	UnitsInStock
Location	5, 0

To connect the Chart to a data source

1. Select the Chart control and at the bottom of the Properties window, select the **Data Source** command. See Properties Window for further details on accessing commands.

   Tip: If the verb is not visible, right-click an empty space in the Properties Window and select **Commands** to display verbs.

2. In the Chart DataSource dialog box that appears, click the **Build** button.
3. In the Data Link Properties window, select **Microsoft Jet 4.0 OLE DB Provider** and click the **Next** button.
4. Click the ellipsis button (...) to browse to the NorthWind database. Click **Open** once you have selected the file.
5. Click the **OK** button to close the window and fill in the Connection String.
6. In the **Query** field, enter the following parameterized SQL query.

SQL Query
`SELECT * FROM Products WHERE CategoryID = <%CategoryID||1%> ORDER BY ProductName`

Note: The chart data cannot be arranged if **ORDER** is not set in both the SQL statements (report and chart statement).

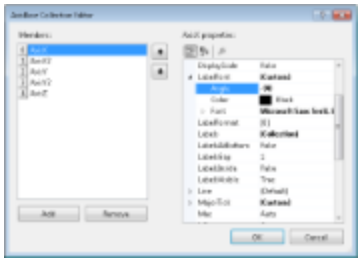
7. Click **OK** to save the data source.

To configure the appearance of the Chart

1. With the chart control selected, go to the Properties window, click the **ChartAreas (Collection)** property and then click the ellipsis button that appears.
2. In the **ChartArea Collection Editor** that appears with defaultArea selected under the members list, click its **Axes (Collection)** property and then click the ellipsis button.
3. In the **AxisBase Collection Editor** that appears, set the following properties.

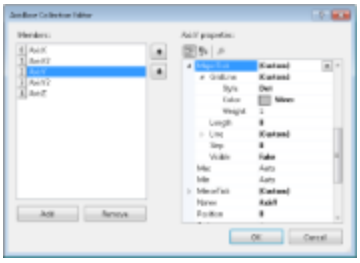
   AxisBase

   1. With AxisX selected under members list, set **Angle** property of **LabelFont** to -90.

   2. Delete text from **Title** property to make sure that the ProductName label (lblProductName) does not overlap.
   3. Click AxisY under the members list and set its **Title** property to Unit Stock.
   4. With AxisY selected under the members list, expand the GridLine section under **MajorTick** property tree view and set its following sub-properties.

Property Name	Property Value
Style	Dot
Color	Silver
Weight	1

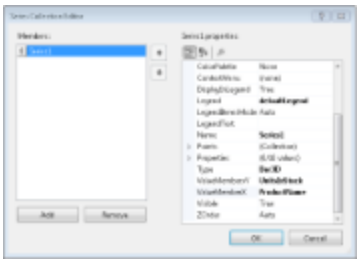
      

   5. Click **OK** to return back to ChartArea collection editor and then click **OK** again to return back to report design surface.
4. With the chart control selected, go to the Properties window, click the **ChartSeries (Collection)** property and then click the ellipsis button that appears.
5. In the **Series Collection Editor** that appears, set the following properties:

   Series

   1. With Series1 selected under the members list, set its following properties:

Property Name	Property Value
ValueMembersY	UnitsInStock
ValueMembersX	ProductName

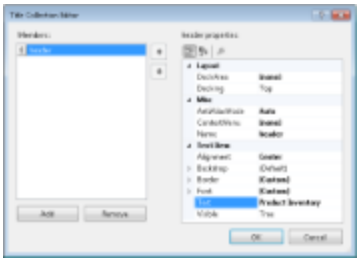


2. Click **OK** to return back to report design surface.

6. With the chart control selected, go to the Properties window, click the **Titles (Collection)** property and then click the ellipsis button that appears.
7. In the **Title Collection Editor** that appears, set the following properties:

Titles

1. With Header selected under the members list, set its **Text** property to Product Inventory.

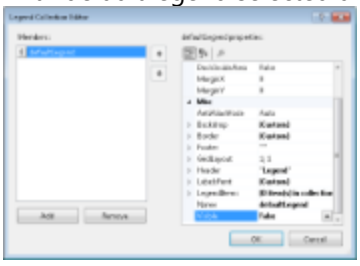


2. Delete/Remove Footer from the members list.
3. Click **OK** to return back to report design surface.

8. With the chart control selected, go to the Properties window, click the **Legends (Collection)** property and then click the ellipsis button that appears.
9. In the **Legend Collection Editor** that appears, set the following properties:

Legends

1. With defaultLegend selected under the members list, set its **Visible** property to False.



2. Click **OK** to return back to report design surface.

To view the report

- Click the preview tab to view the report at design time.

OR

- Open the report in the Viewer. See Windows Forms Viewer for further information.

Web

This section contains the following walkthroughs that fall under the Web category.

Document Web Service
: This walkthrough describes how to set up a simple web service that returns an ActiveReports document.

Document Windows Application
: This walkthrough describes how to set up a Windows client application for the ActiveReports Document Web Service

> ⚠ **Important**: In order to consume Web services in your Windows applications, you must set permissions to allow the ASP.NET user to consume the services. Ask your server administrator for help with this.

Document Web Service

With ASP.NET and ActiveReports, you can set up a Web Service that returns a report document which can be shown in a report viewer control.

This walkthrough illustrates how to create a Web Service that returns the contents of an ActiveReports as a byte array.

This walkthrough is split up into the following activities:

- Creating an ASP.NET Web Service project
- Adding code to create the Web Method
- Testing the Web Service
- Publishing the Web Service
- Creating a virtual directory in IIS

> **Note:** For the information on how to connect your report to data and how to create the report layout, please see Basic Data Bound Reports for a section report.

When you have completed this walkthrough, you will have a Web Service that returns the contents of an ActiveReports as a byte array.

To create an ASP.NET Web Service project

1. From the **File** menu, select **New Project.**
2. In the **New Project** dialog that appears, select **ASP.NET Web Service Application**.
3. Change the name of the project.
4. Click **OK** to open the new project in Visual Studio.

To write the code to create the Web Method

1. On the **Service.vb** or **Service.cs** tab is the code view of the Service.asmx file.
2. Replace the existing WebMethod and HelloWorld function with the following code.

The following code demonstrates how you create the Web Method for a section report.

Visual Basic.NET code. REPLACE the existing WebMethod and function with this code.

```
<WebMethod( _Description:="Returns a products report grouped by category")> _
Public Function GetProductsReport() As Byte()
     Dim rpt As New rptProducts()
     rpt.Run()
     Return rpt.Document.Content
End Function
```

C# code. REPLACE the existing WebMethod and function with this code.

```
[WebMethod(_Description = "Returns a products report grouped by category")]
public byte[] GetProductsReport()
{
    rptProducts rpt = new rptProducts();
    rpt.Run();
    return rpt.Document.Content;
}
```

To test the Document Web Service

1. Press **F5** to run the project. The Service page appears in your browser.
2. In the list of supported operations at the top, click **GetProductsReport**.
3. Click the **Invoke** button to test the Web Service operation.
4. If the test is successful, you will see the binary version of the contents of rptProducts.

To publish the Document Web Service

1. In the Solution Explorer, right-click the project name and select **Publish**.
2. In the Publish Web window that appears, enter locahost in the **Service URL field** and "*SiteName*"/WebService in the **Site/application** field.

   > **Note**: Get the *SiteName* from the **Internet Information Services Manager**.

3. Select the **Mark an IIS application on destination** option and click the **OK** button.

To check the configuration in IIS

1. Open **Internet Information Services Manager**.
2. In the **Internet Information Services Manager** window that appears, expand the tree view in the left pane until you see the Web Service you had added in the steps above.
3. Right-click the Web Service select **Manage Application** then **Browse**.
4. In the browser that appears, go to the Address bar and add \Service1 to the url.

For information on consuming the Document Web Service in a viewer, see Document Windows Application.

Document Windows Application

In ActiveReports, you can use a Web Service that returns the content of a Section report to show in the Windows Forms viewer control.

This walkthrough illustrates how to create a Windows client application that returns the content of a Section report in the Windows Forms viewer.

This walkthrough builds on the Document Web Service walkthrough and is split up into the following activities:

- Creating a Visual Studio project
- Adding the ActiveReports Windows Forms viewer control to the form
- Adding a reference to a Web service to the project
- Displaying the content returned by the Document Web Service in the viewer
- Running the project

> **Note:** Refer to Document Web Service to set up Web service that returns an ActiveReports document.

To create a Visual Studio project

1. From the **File** menu, select **New**, then **Project**.
2. In the Templates section of the New Project dialog, select **Windows Application**.
3. Change the name of the application to **ARDocumentClient**.
4. Click **OK** to open the project.

To add the Viewer control

1. From the Visual Studio toolbox, drag the ActiveReports **Viewer** control onto the form.
2. Change the **Dock** property for the viewer control to **Fill**, and resize the form to accommodate a report.

To add a web reference

To add a reference to a web service in Visual Studio that is compatible with the .NET Framework Web service version

1. From the **Project** menu, select **Add Service Reference**.
2. In the **Add Service Reference** window that appears, click the **Advanced** button.
3. In the **Service Reference Settings** window that appears, click **Add Web Reference** button.
4. From the **Project** menu, select **Add Web Reference**.
5. Type in the address of the .asmx file for the ActiveReports Document Web Service you created in the previous walkthrough. For example: **http://localhost/ARDocumentWS/Service.asmx**
6. Click the **Add Reference** button when the Web Service is recognized.

To add a reference to a web service in Visual Studio

1. From the **Project** menu, select **Add Service Reference**.
2. In the **Add Service Reference** that appears, type in the address of the .asmx file for the ActiveReports Document Web Service you created in the previous walkthrough. For example: **http://localhost/ARDocumentWS/Service.asmx**
3. Click the **Go** button, and then click the **OK** button when the Web Service is recognized.

To display the content returned by the Document Web Service in the viewer

To display the report content

1. Double-click Form1 to create an event-handling method for the Form1_Load event.
2. Add code to the handler to display the document Web service content in the viewer.

The following example shows what the code for the method looks like.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the Form Load event.

```
Dim ws As New localhost.Service
Me.Viewer1.Document.Content = ws.GetProductsReport()
```

To write the code in C#

C# code. Paste INSIDE the Form Load event.

```
localhost.Service ws = new localhost.Service();
this.viewer1.Document.Content = ws.GetProductsReport();
```

To display the report content

1. Double-click on Form1 to create an event-handling method for the Form1_Load event.
2. Add code to the handler to display the document Web service content in the viewer.

The following example shows what the code for the method looks like.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste INSIDE the Form Load event.

```
Dim ws As New ServiceReference1.ServiceSoapClient()
Me.Viewer1.Document.Content = ws.GetProductsReport()
```

To write the code in C#

C# code. Paste INSIDE the Form Load event.

```
ServiceReference1.ServiceSoapClient ws = new ServiceReference1.ServiceSoapClient();
this.viewer1.Document.Content = ws.GetProductsReport();
```

To update the app.config file

> **Note:** You need to update the app.config file if you added the Service Reference to Visual Studio project in the previous section.

1. In the Solution Explorer, open the app.config file.
2. In the tag <binding name = "ServiceSoap"...>, set **maxBufferSize** and **maxReceivedMessageSize** to some large number, for example, 200500.
3. In the next tag <readerQuotas...>, set **maxArrayLength** to some large number, for example, 60500.

To run the project

Press **F5** to run the project.

Common Walkthroughs

Common walkthroughs cover scenarios to introduce the key features of page and section reports. Learn about different

page and section report walkthroughs categorized as follows.

Professional
: This section contains the walkthroughs explaining features that are part of the ActiveReports Professional Edition.

Web
: This section contains the walkthroughs that explain how to create a simple web service for each scenario and how to create a Windows client application for each web service.

Professional

This section contains the following walkthroughs that fall under the Professional category.

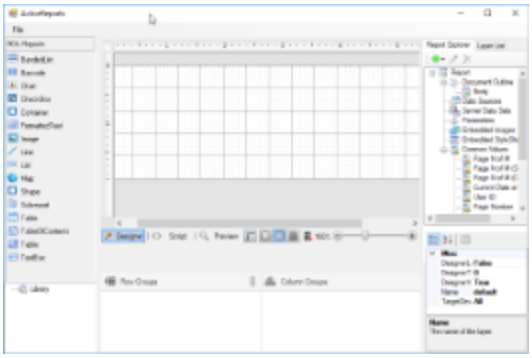
Creating a Basic End User Designer
: This walkthrough demonstrates how to set up a basic end-user report designer on a Windows Forms application.

Customizing the HTML Viewer UI
: This walkthrough demonstrates how to customize the HTML Viewer interface using JQuery methods.

Creating a Basic End User Report Designer (Pro Edition)

This walkthrough illustrates how to set up a basic End-User Report Designer on a Windows Forms application in the Professional Edition of ActiveReports.

At the end of this walkthrough, the End-User Report Designer appears like the following image.



Adding the Designer control to the Form

You will add the Designer that could only edit and preview a report file.

1. Create a new Windows Forms Application project.
2. In the **Name** field, rename the file to CustomEUD and click **OK**.
3. Install **GrapeCity.ActiveReports** package from **nuget.org** to make the ActiveReports 14 toolbox available in Visual Studio.
4. Install **GrapeCity.ActiveReports.Design.Win** package to make the **Designer** control available in the toolbox.
5. Select a Form and go to the Properties Window to change the **Name** property to **frmMain** and the **Text** property to **ActiveReports**.
6. Resize the Form so that you can accommodate the controls listed further.
7. From the Visual Studio toolbox, drag the Designer control onto the Form and rename it to **designer**.
8. From the Visual Studio toolbox, drag the Toolbox control onto the Form and rename it to **toolbox**.

9. To attach the toolbox control to the designer control, in the Solution Explorer, right-click Form1.cs and select **View Code.**
10. Add the following code (marked in bold) after the InitializeComponent method.

C# code. Paste AFTER the InitializeComponent method

```
public frmMain()
{
  InitializeComponent();
  designer.Toolbox = toolbox;
}
```

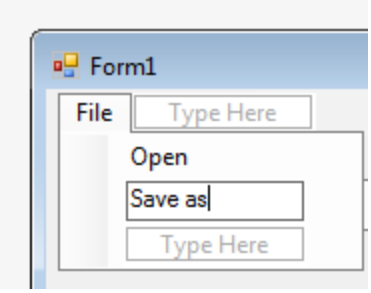
11. At the top of the code view, add a using directive.

C# code. Paste at the top of the Form1 code view

```
using GrapeCity.ActiveReports.Design;
```

Loading and/or saving the report file

1. From the Visual Studio **Menus & Toolbars** toolbox group, drag the MenuStrip control onto the Form.
2. Create the following structure for the MenuStrip control: **File > Open**, **File > Save as**.



3. On the Form, double-click the **Open** menu item and paste the following code (marked in bold) into the openToolStripMenuItem_Click handler.

C# code. Paste INSIDE the openToolStripMenuItem_Click handler

```
private void openToolStripMenuItem_Click(object sender, EventArgs e)
        {
            OpenFileDialog openFileDialog = new OpenFileDialog();
            var dialogResult = openFileDialog.ShowDialog();
            if (dialogResult == System.Windows.Forms.DialogResult.OK)
            {
                designer.LoadReport(new
System.IO.FileInfo(openFileDialog.FileName));
            }
        }
```

4. On the Form, double-click the **Save as** menu item and paste the following code (marked in bold) into the saveAsToolStripMenuItem_Click handler.

C# code. Paste INSIDE the saveAsToolStripMenuItem_Click handler

```
private void saveAsToolStripMenuItem_Click(object sender, EventArgs e)
```

```
        {
            SaveFileDialog saveFileDialog = new SaveFileDialog();
            saveFileDialog.Filter = GetSaveFilter();
            var dialogResult = saveFileDialog.ShowDialog();
            if (dialogResult == System.Windows.Forms.DialogResult.OK)
            {
               designer.SaveReport(new
System.IO.FileInfo(saveFileDialog.FileName));
            }
        }
```

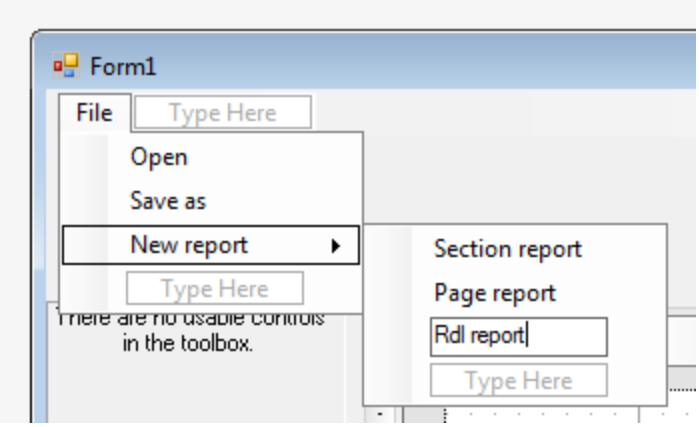
5. After the saveAsToolStripMenuItem_Click handler code, add the code for the GetSaveFilter method as follows.

C# code. Paste AFTER the saveAsToolStripMenuItem_Click handler

```
private string GetSaveFilter()
 {
     switch (designer.ReportType)
     {
        case DesignerReportType.Section:
           return "Section Report Files (*.rpx)|*.rpx";
        case DesignerReportType.Page:
           return "Page Report Files (*.rdlx)|*.rdlx";
        case DesignerReportType.Rdl:
           return "RDL Report Files (*.rdlx)|*.rdlx";
        default:
           return "RDL Report Files (*.rdlx)|*.rdlx";
     }
 }
```

Creating a new report based on a chosen type

1. Create the following structure for the MenuStrip control: **File > New report > Section report**; **File > New report > Page report; File > New report > Rdl report**.



2. Double-click the **Section report** MenuStrip item and add the following code (marked in bold) into the

sectionToolStripMenuItem_Click handler.

C# code. Paste INSIDE the sectionToolStripMenuItem_Click handler

```
private void sectionToolStripMenuItem_Click(object sender, EventArgs e)
  {
   designer.NewReport(DesignerReportType.Section);
  }
```

3. Double-click the **Page report** MenuStrip item and add the following code (marked in bold) into the pageReportToolStripMenuItem_Click handler.

C# code. Paste INSIDE the pageReportToolStripMenuItem_Click handler

```
private void pageReportToolStripMenuItem_Click(object sender, EventArgs e)
  {
    designer.NewReport(DesignerReportType.Page);
  }
```

4. Double-click the **Rdl report** MenuStrip item and add the following code (marked in bold) into the rdlReportToolStripMenuItem_Click handler.

C# code. Paste INSIDE the rdlReportToolStripMenuItem_Click handler

```
private void rdlReportToolStripMenuItem_Click(object sender, EventArgs e)
  {
   designer.NewReport(DesignerReportType.Rdl);
  }
```

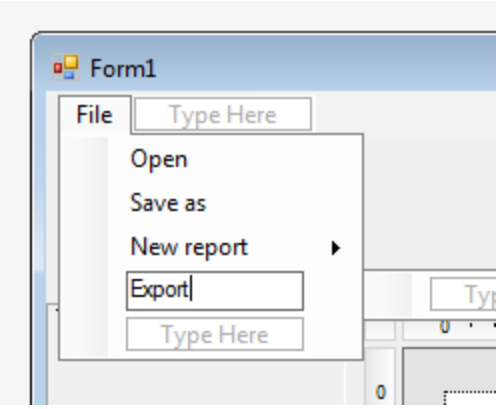
Adding the Export option

1. Install packages from **nuget** as follows:
   i) Go to **Tools** > **Nuget Package Manager** > **Manage Nuget Packages for Solution**...
   ii) Browse the following package and click Install.
   GrapeCity.ActiveReports.export.Pdf
2. At the top of the code view, add using statements.

C# code. Paste at the top of the Form1 code view

```
using GrapeCity.ActiveReports.Export.Pdf.Page;
using GrapeCity.ActiveReports.Rendering.IO;
using GrapeCity.ActiveReports;
using System.IO;
```

3. In the MenuStrip control, add the Export menu item to the **File** menu.



4. In the Properties Window, set the **Enabled** property to **False**. This enables the Export menu item to be displayed in the Preview mode only.
5. To enable the Export menu item to be displayed for page and rdl reports only, add the following code (marked in bold) after the InitializeComponent method.

C# code. Paste AFTER the InitializeComponent method

```
public frmMain()
        {
            InitializeComponent();
            designer.Toolbox = toolbox;
            designer.ActiveTabChanged += designer_ActiveTabChanged;
        }

void designer_ActiveTabChanged(object sender, EventArgs e)
        {
            exportToolStripMenuItem.Enabled = designer.ActiveTab ==
DesignerTab.Preview && designer.ReportType != DesignerReportType.Section;
        }
```

6. On the Form, double-click the Export item and add the following code (marked in bold) to the exportToolStripMenuItem_Click handler.

C# code. Paste INSIDE the exportToolStripMenuItem_Click handler

```
private void exportToolStripMenuItem_Click(object sender, EventArgs e)
        {
            SaveFileDialog saveFileDialog = new SaveFileDialog();
            saveFileDialog.Filter = "Pdf (*.pdf)|*.pdf";
            var dialogResult = saveFileDialog.ShowDialog();
            if (dialogResult == System.Windows.Forms.DialogResult.OK)
            {
                var pdfRe = new PdfRenderingExtension();
                var msp = new MemoryStreamProvider();
                (designer.Report as PageReport).Document.Render(pdfRe, msp);
                using (var stream = msp.GetPrimaryStream().OpenStream())
               using (var fileStream = new FileStream(saveFileDialog.FileName,
```

```
FileMode.Create, FileAccess.Write))
                {
                    stream.CopyTo(fileStream);
                }
                MessageBox.Show("Export is done");
            }        }
```

For more information on export filters, rendering extensions and their settings, refer to the Exporting section of the ActiveReports User Guide.

Adding other controls to the Form

1. From the Visual Studio toolbox, drag the following controls onto the Form.

Control	Name	Property Value
ReportExplorer	arReportExplorer	ReportDesigner = designer This binds the ActiveReports Designer to the ReportExplorer control. Resize and move as necessary.
LayerList	arLayerList	ReportDesigner = designer This binds the ActiveReports Designer to the LayerList control. Resize and move as necessary.
PropertyGrid	arPropertyGrid	Resize and move as necessary.
GroupEditor	arGroupEditor	ReportDesigner = designer This binds the ActiveReports Designer to the GroupEditor control. Resize and move as necessary.
ReportsLibrary	arReportsLibrary	ReportDesigner = designer This binds the ActiveReports Designer to the ReportsLibrary control. Resize and move as necessary.

2. On the Form, select the Designer control.
3. In the Properties Window, set the **PropertyGrid** property of the Designer control to arPropertyGrid. This binds the ActiveReports Designer to the Property Grid control.

Viewing the End User Report Designer

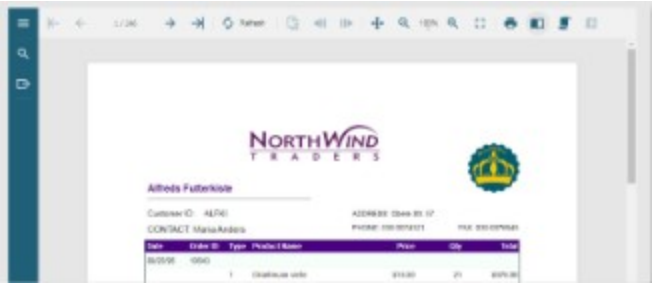
Press F5 to run the project. The **End User Report Designer** opens with an RDL Report.

For information on how you can customize the End User Report Designer and more, refer to the End User Designer product sample.

Customizing the WebViewer UI

You can customize the WebViewer interface using JQuery methods. WebViewer control adds JQuery library in page scripts. Use the code in this walkthrough to add a button on the toolbar and add a client side PDF export implementation.

When you complete this walkthrough you get a WebViewer that looks similar to the following at run time.



Load an ActiveReport to the Web application

1. Create a new Visual Studio **ASP.NET Web Forms** application.
2. Install **GrapeCity.ActiveReports.Web** package. Go to **Tools > Nuget Package Manager > Manage Nuget Packages for Solution...**, browse for the package and click **Install**.
3. In Solution Explorer, right-click the project and select **Add > New Item**.
4. Select WebForm and click **Add**.
5. Go to the **Design** view of the newly added WebForm.aspx and drag and drop the WebViewer control to the WebForm designer. The default viewer type is **HTMLViewer**.
6. Load a report in the WebViewer by setting the **ReportName** property.

> **Note**: You may load any report, section or page in the HTMLViewer viewer type of WebViewer. See Getting Started with the WebViewer for information on loading a report.

Add the jQuery library to the Web application project

In the Source view of the **WebForm.aspx** file, add the following code.

Add this code after the <head> tag

```
<script src="https://code.jquery.com/jquery-2.1.4.min.js"></script>
```

Access the WebViewer view model

The HTML WebViewer is created using the MVVM pattern that provides a view model which does not depend on the UI. The code can access the Viewer's view model and bind the custom UI to it by using well-documented properties and methods. For MVVM support, the code can use knockout.js which is the standard MVVM library for JavaScript. Knockout.js provides declarative bindings, observable objects and collections in HTML markup. See Using Javascript with the HTML Viewer for more information.

Follow the steps below to access the ViewModel.

1. In the Source view of the **WebForm.aspx** file, add a <script> tag.
2. Add the following Javascript code for document's **Onload** event handler and WebViewer's **Loaded** event handler that gets fired when the UI is rendered on the Html Page:

Paste the code into .aspx source

```
<script>
function viewer_loaded()
  {
```

```
  };
function document_onload()
  {
  };
</script>
...
<body onload="document_onload()">
```

3. Add the following Javascript code inside the **viewer_loaded** event handler to access WebViewer's view model:

Paste the code into .aspx source

```
function viewer_loaded() {
            var viewModel = GetWebViewer('WebViewer1');
        };
```

4. Add the following Javascript code inside the **document_onload** event handler to bind WebViewer's Loaded event to client side viewer_loaded event:

Paste the code into .aspx source

```
function document_onload() {
            $('#WebViewer1').ready(viewer_loaded);
        };
```

Add a button to the WebViewer toolbar

In the Source view of the WebForms.aspx file, add the following Javascript code inside the **viewer_loaded** event handler to access the WebViewer toolbar. Lets add the custom button in the toolbar - an export button and add PDF export functionality to it.

Paste the code into .aspx source

```
function viewer_loaded() {
            var viewModel = GetWebViewer('WebViewer1');

            var pdfExportButton = {
                key: '$pdfExportButtonKey',
                text: 'To PDF',
            iconCssClass: 'mdi mdi-file-pdf',
                enabled: true,
                action: function (item) {
                    console.log('Export to PDF function works here');
                },
                onUpdate: function (arg, item) {
                    console.log(The Viewer UI was updated, check/update button state
here');
                }
            };

            viewModel.toolbar.desktop.addItem(pdfExportButton);
```

```
        };
```

The complete code for WebForm1.aspx in the Source view is as shown:

Paste the code into .aspx source

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="WebForm1.aspx.cs"
Inherits="WebApplication1.WebForm1" %>
<%@ Register assembly="GrapeCity.ActiveReports.Web"
namespace="GrapeCity.ActiveReports.Web" tagprefix="ActiveReportsWeb" %>

<!DOCTYPE html>
<html xmlns="https://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body onload="document_onload()">
    <form id="form1" runat="server">
        <div>
            <ActiveReportsWeb:WebViewer ID="WebViewer1" runat="server" height="466px"
width="667px" ReportName="AllCustomers.rdlx">
            </ActiveReportsWeb:WebViewer>
        </div>

    </form>
    <script src="https://code.jquery.com/jquery-2.1.4.min.js">
    </script>
    <script>
        function viewer_loaded() {
            var viewModel = GetWebViewer('WebViewer1');

            var pdfExportButton = {
                key: '$pdfExportButtonKey',
                text: 'To PDF',
                iconCssClass: 'mdi mdi-file-pdf',
                enabled: true,
                action: function (item) {
                    console.log('Export to PDF function works here');
                },
                onUpdate: function (arg, item) {
                    console.log('Something in viewer was updated, check/update button
state here');
                }
            };

            viewModel.toolbar.desktop.addItem(pdfExportButton);
```

```
        };

        function document_onload() {
            $('#WebViewer1').ready(viewer_loaded);
        };
    </script>
</body>
</html>
```

To remove a button from the viewer's UI

Paste the code into .aspx source

```
function viewer_loaded() {
            var viewModel = GetWebViewer('WebViewer1');
            ...

            viewModel.toolbar.desktop.removeItem($newButtonKey); //(key of the button)
        };
```

Note:

- Replace 'WebViewer1' in the code snippets above, with the actual ID of the WebViewer control in your application.
- In case you provide report name which contains special symbols (like backslash '\'), e.g. webViewer.ReportName="Folder\Report.rdlx", you need to update the web.config file to allow such characters. Otherwise, "Report not found" error occurs. Please see Troubleshooting on resolving this issue.

Web

This section contains the following walkthroughs that fall under the Web category.

DataSet Web Service
: This walkthrough describes how to set up a simple web service that returns a dataset.

DataSet Windows Application
: This walkthrough describes how to set up a Windows client application for the dataset Web Service.

Important: In order to consume Web services in your Windows applications, you must set permissions to allow the ASP.NET user to consume the services. Ask your server administrator for help with this.

DataSet Web Service

With ASP.NET, you can set up a Web Service that returns a dataset to use in ActiveReports. This walkthrough illustrates how to create one.

This walkthrough is split into the following activities:

- Creating an ASP.NET Web Service project
- Adding code to create the Web method
- Testing the Web service
- Publishing the Web service
- Creating a virtual directory in IIS

Note: For the information on how to connect your report to data and how to create the report layout, please see Single Layout Reports (for a page report) or Basic Data Bound Reports (for a section report).

To create an ASP.NET Web Service project

1. From the **File** menu, select **New Project.**
2. In the **New Project** dialog that appears, select **ASP.NET Web Application** to create a empty web application.
3. Change the name of the project.
4. Click **OK** to open the new project in Visual Studio.

To create the Web Method

1. From the **Project** menu, select **Add New Item**.
2. In the **Add New Item** dialog that appears, select **Web Service (asmx)** and change the name of the web service.

In the *WebService*, replace the existing <WebMethod()> _ and HelloWorld function with code like the following.

To write the code in Visual Basic.NET

Visual Basic.NET code. Paste OVER the existing WebMethod.

```
Private connString As String
<WebMethod(Description:="Returns a DataSet containing all Products")> _
Public Function GetProduct() As Data.DataSet
connString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=[User
Folder]\Samples14\Data\Nwind.mdb"
Dim adapter As New Data.OleDb.OleDbDataAdapter("select * from products", connString)
Dim ds As New Data.DataSet()
adapter.Fill(ds, "Products")
Return ds
End Function
```

To write the code in C#

C# code. Paste OVER the existing WebMethod.

```
private static string connString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source = [User
Folder]\\Samples14\\Data\\nwind.mdb";
[WebMethod(Description="Returns a DataSet containing all Products")]
public Data.DataSet GetProduct()
{
   System.Data.OleDb.OleDbDataAdapter adapter;
   System.Data.DataSet ds;
   adapter = new System.Data.OleDb.OleDbDataAdapter("select * from products",
connString);
```

```
    ds = new System.Data.DataSet();
    adapter.Fill(ds, "Products");
    return ds;
}
```

To test the Web Service

1. Press **F5** to run the project.
2. If the Debugging Not Enabled dialog appears, select the option that enables debugging and click **OK** to continue.
3. In the list of supported operations, click the **GetProduct** link. (The description string from the code above appears below the link.)
4. Click the **Invoke** button to test the Web Service operation.
5. If the test is successful, a valid XML schema of the Northwind products table displays in a new browser window.
6. Copy the URL from the browser for use in the Web Reference of your DataSet Windows Application.

To publish the Web Service

1. In the Solution Explorer, right-click the project name and select **Publish**.
2. In the Publish Web window that appears, select Custom option to create a custom profile and click **OK**.
3. Enter *localhost* in the **Server field** and "*SiteName*"/*WebServiceName* in the **Site name** field.

   > **Note**: Get the *SiteName* from the **Internet Information Services Manager**.

4. Click the **Publish** button.

To check the configuration in IIS

1. Open **Internet Information Services Manager**.
2. In the **Internet Information Services Manager** window that appears, expand the tree view in the left pane until you see the Web Service you had added in the steps above.
3. Right-click the Web Service select **Manage Application** then **Browse**.
4. In the browser that appears, go to the Address bar to add ***WebServiceName.asmx*** to the url and press Enter.

For information on consuming the DataSet Web Service in an ActiveReport, see DataSet Windows Application.

DataSet Windows Application

You can use a Web Service that returns a dataset as the data source for your reports in Windows applications. This walkthrough illustrates how to create a Windows client application that uses the dataset Web Service as the data source for an ActiveReports.

This walkthrough builds on the DataSet Web Service walkthrough and is split up into the following activities:

- Adding a reference to a Web service to the project
- Setting the report data source to the one returned by the Web service

> **Note:** For the information on how to connect your report to data and how to create the report layout, please see Single Layout Reports (for a page report) or Basic Data Bound Reports (for a section report).

To add a reference to a web service in Visual Studio

1. From the **Project** menu, select **Add Service Reference**.

2. In the **Add Service Reference** window that appears, type in the address of the virtual directory you created in the previous walkthrough. You can get the address by running the project from the previous walkthrough and copying the url from the address in the browser. (It will look something like http://localhost:####/DataSetWS/Service.asmx where #### is the port number.)
3. Click the **Go** button, and then click the **OK** button when the Web Service is recognized.

To set the report data source to the one returned by the Web service

To set the report data source (for Visual Studio compatible with .NET Framework Web service version)

1. Double-click the gray area below the report. This creates an event-handling method for the ReportStart event.
2. Add code to the handler to use the web service dataset in the report.

   The following example shows what the code for the method looks like.

   To write the code in Visual Basic.NET

   Visual Basic.NET code. Paste INSIDE the ReportStart event

```
Dim ws As New localhost.Service
Dim ds As DataSet()= ws.GetProduct()
Me.DataSource = ds
Me.DataMember = "Products"
```

   To write the code in C#

   C# code. Paste INSIDE the ReportStart event.

```
localhost.DataSetWS ws = new localhost.Service;
DataSet ds = ws.GetProduct();
this.DataSource = ds;
this.DataMember = "Products";
```

To set the report data source

1. Double-click the gray area below the report. This creates an event-handling method for the ReportStart event.
2. Add code to the handler to use the web service dataset in the report.

   The following example shows what the code for the method looks like.

   To write the code in Visual Basic.NET

   Visual Basic.NET code. Paste INSIDE the ReportStart event.

```
Dim ws As New ServiceReference1.ServiceSoapClient()
Dim ds As DataSet = ws.GetProduct()
Me.DataSource = ds
Me.DataMember = "Products"
```

   To write the code in C#

   C# code. Paste INSIDE the ReportStart event.

```
ServiceReference1.ServiceSoapClient ws = new ServiceReference1.ServiceSoapClient();
```

```
DataSet ds = ws.GetProduct();
this.DataSource = ds;
this.DataMember = "Products";
```

To update the app.config file

> **Note:** You need to update the app.config file if you added the Service Reference to the Visual Studio project in the previous section.

1. In the Solution Explorer, open the app.config file.
2. In the tag <binding name = "ServiceSoap"...>, set **maxBufferSize** and **maxReceivedMessageSize** to some large number, for example, 200500.
3. In the next tag <readerQuotas...>, set **maxArrayLength** to some large number, for example, 60500.

Troubleshooting

If you run into an issue while using ActiveReports, you will probably find the solution within this section. Click any short description below to drop down the symptoms, cause, and solution. Or click a link to another section of the troubleshooting guide.

General Troubleshooting

Error appears on exporting Page/RDL reports from JSViewer run on ASP.NET Core MVC application

Symptoms: "Export report error" appears on exporting Page/RDL report from JSViewer when run through ASP.NET Core MVC applications.

Cause: ASP.NET Core MVC has undergone some changes that disable the synchronous server operations. See this for more information.

Solution: Add following content in Startup.cs to turn on the synchronous operations.

Startup.cs

```
services.Configure<IISServerOptions>(options =>
{
    options.AllowSynchronousIO = true;
});
```

Error appears on using parameterized queries with OLE DB provider in .NET core applications

Symptoms: Error is thrown on using parameterized queries with OLE DB provider in .NET Core applications.

Cause: The number of query parameters specified in the dataset do not match the parameters used in the SQL query for dataset. Note that this works perfectly well for OLE DB provider from Full .NET Framework.

Solution: The number of query parameters specified in the SQL query should be same as number of query parameters in dataset.

Report menu does not appear in the main menu of Visual Studio 2019

Symptoms: When a report is opened in Visual Studio 2019, Report menu does not appear in the main menu.

Cause: Due to new behavior relating to extensions in Visual Studio 2019, Report menu is removed as main menu.

Solution: Go to **Extensions** menu. You can see that the Report menu is available as submenu.

References missing from Visual Studio Add Reference dialog

Symptoms: When you try to add references to your project, only a few of the ActiveReports references are available.

Cause: The project's target framework is set to an old version of the .NET framework that does not support the new assemblies.

Solution:

1. In the Solution Explorer, right click the project and choose Properties.
2. On the Application tab in C# projects (or the Compile tab, then the Advanced Compile Options button in Visual Basic projects), drop down the Target framework box and select .NET Framework 4.6.2.

Errors after installing a new build

Symptoms: When you open a project created with a previous build of ActiveReports after installing a new build, there are errors related to being unable to find the previous build.

Cause: Visual Studio has a property on references called Specific Version. If this property is set to True, the project looks for the specific version that you had installed when you created the report, and throws errors when it cannot find it.

Solution: For each of the ActiveReports references in the Solution Explorer, select the reference and change the Specific Version property to False in the Properties Window.

The project does not work if Integrated Managed Pipeline Mode is enabled

Symptoms: The web project does not work in the application pool if Integrated Managed Pipeline Mode is enabled.

Cause: The application configuration is incorrect for being used in Integrated mode.

Solution: Migrate the application configuration. Here is a sample command.

Paste the following on the command line.

```
"%SystemRoot%\system32\inetsrv\appcmd migrate config YourWebSite/"
```

GrapeCity.ActiveReports.Extensiblity.dll is added to the list of unused dlls in Visual Studio

Symptoms: GrapeCity.ActiveReports.Extensiblity.dll is added to the references folder when the viewer is dragged and dropped on the form, however it is present in the list of unused dll in Visual Studio.

Cause: GrapeCity.ActiveReports.Extensiblity.dll is used internally for certain features in ActiveReports and is present in the Global Assembly Cache. Therefore, Visual Studio can resolve the dependency and cannot find the direct references due to which it gets listed as a unused dll in Visual Studio.

Reports are not associated with the designer in Visual Studio when adding ActiveReports to a TFS-bound project

Symptoms: When adding ActiveReports to a Web site project that is bound to TFS (where reports are added to the **App_Code** folder), the report does not open in the designer in Visual Studio.

Cause: The **FileAttributes.xml** file that contains attribute information to associate ActiveReports files with the Designer is usually loaded and maintained in memory when a new ActiveReports file is added. However, if a Web site is bound to TFS, the FileAttributes.xml file is not maintained in memory. As a result, Visual Studio treats all the newly added files as normal code files.

Solution: Add the newly added reports to **FileAttributes.xml** manually.

1. From the **Website** menu, select **Add New Item**.
2. Select **ActiveReports 14 Section Report (code-based)** and click **OK**.
3. Close the project.
4. From Windows Explorer, open the **FileAttributes.xml** file in an editor and add the new ActiveReports file, setting the subtype to **Component**, using code like the following.

   Note: The FileAttributes.xml is located at C:\Documents and Settings\[username]\Local Settings\Application Data\Microsoft\WebsiteCache\[WebSite1]\ (Windows XP), or at C:\Users\[username]\AppData\Local\Microsoft\WebsiteCache\[WebSite1]\ (Windows 7).

   XML code. Paste inside FileAttributes.xml

   ```
   <?xml version="1.0" encoding="utf-16" ?>
   <DesignTimeData>
      <File RelativeUrl="App_Code/NewActiveReport1.cs" subtype="Component" />
      <File RelativeUrl="Default.aspx.cs" subtype="ASPXCodeBehind" codebehindowner="Default.aspx" />
      <File RelativeUrl="Default.aspx" subtype="ASPXCodeBehind" />
   </DesignTimeData>
   ```

5. Save the **FileAttributes.xml** file.
6. Reopen the Web site project.

The SystemNotSupportedException occurs when running ActiveReports with scripts on .NET Framework 4.0

Symptoms: The SystemNotSupportedException occurs when running ActiveReports with scripts on .NET Framework 4.0.

Cause: This exception occurs because of the CAS policy, which is obsolete in the .NET Framework 4.0.

Solution: To resolve this issue, the configuration file needs to be updated. To do this, in the Solution Explorer, open the app.config file (for Windows Forms applications) or the Web.config file (for ASP.NET Web applications) and add the following code.

(Windows Forms Applications) XML code. Paste inside the app.config file

```
<configuration>
  <runtime>
    <NetFx40_LegacySecurityPolicy enabled="true"/>
  </runtime>
</configuration>
```

(ASP.NET Web Applications) XML code. Paste inside the Web.config file

```
<system.web>
   <trust legacyCasModel="true"/>
</system.web>
```

An Exception occurs on previewing reports connecting Microsoft Access OLE DB provider in a 64-bit system

Symptoms: "System.InvalidOperationException: The 'Microsoft.Jet.OLEDB.4.0' provider is not registered on the local machine." occurs on previewing reports connecting to Microsoft.Jet.OLEDB.4.0 provider on a 64-bit operating system.

Cause: The Microsoft Access OLE DB provider, Microsoft.Jet.OLEDB.4.0, is not compatible with 64 bit, so it fails on a 64-bit operating systems.

Solution: To avoid this, you have two options.

1. (Preferred) Change the OLE DB Provider to **Microsoft.ACE.OLEDB.12.0**.

Note that both 32-bit and 64-bit version of Microsoft.ACE.OLEDB.12.0 should be available in your machine.

2. Change the project settings to use only **32 bit**.

1. With the project open in Visual Studio, from the **Project** menu, select Project Properties.
2. In the page that appears, select the **Compile** tab in a VB project, or the **Build** tab in a C# project.
3. Scroll to the bottom of the page and click the **Advanced Compile Options** button in VB, or skip this step in C#.
4. Drop down the **Target CPU** list in VB, or **Platform target** in C#, (set to use AnyCPU by default) and select **x86**.
5. Click **OK** to save the changes, or skip this step in C#.

The printing thread dies before the report finishes printing

Symptoms: The printing thread dies before the report is printed.

Cause: If printing is done in a separate thread and the application is shut down right after the print call, the separate thread dies before the report is printed.

Solution: Set the usePrintingThread parameter of the Print() method to False to keep the printing on the same thread. This applies to all Page reports, RDL reports and Section reports.

1. In the project where you call the Print method, add a reference to the GrapeCity.ActiveReports.Viewer.Win assembly.
2. At the top of the code file where you call the Print method, add a using directive (Imports for VB) for **GrapeCity.ActiveReports**.
3. Call the Print method with the usePrintingThread parameter (the third parameter) set to false with code like the following.

C# code.

```
document.Print(false, false, false);
```

Visual Basic code.

```
document.Print(False, False, False)
```

Exception thrown when using Viewer.Print to print a report

Symptoms: An exception is thrown when the Viewer.Print method is used to print a report.

Cause: Print method was called before the page was loaded completely.

Solution: Use the Viewer.Print method in the **LoadCompleted ('LoadCompleted Event' in the on-line documentation)** event.

Error on previewing Server reports

Symptoms: When you try to preview Server reports created in previous ActiveReports versions in ActiveReports 14 designer, the error 'Server report is not supported' occurs.

Cause: The Server credentials are present in report definition and ActiveReports Server is not supported in ActiveReports 14.

Solution: Edit the report in any text editor and remove the <Custom Property> tag that contains the Server name and URL.

Some Data Providers are not available in DataSource Editor in .NET Core applications

Symptoms: Microsoft Data Providers such as Sql Client Provider, OleDb Provider, and Odbc Provider are not available in DataSource Editor in Designer in .NET Core 3.1 applications.

Cause: This is because of the references to older versions of Microsoft compatibility pack, etc.

Solution: Update the following references to versions:

- Microsoft.Windows.Compatibility 3.0.0 -> 3.1.0
- System.Data.SqlClient 4.7.0 -> 4.8.0
- System.Text.Encoding.CodePages 4.6.0 -> 4.7.0

Code generation error appears with code-based report applications

Symptoms: The Code generation error appears when working with a code-based application in Visual Studio.

Cause: This is because of the cache problems in Visual Studio.

Solution: Use one of the solutions described below.

- Clean Visual Studio cache.
  - Remove the folder C:\Users\[user name]\AppData\Local\Microsoft\VisualStudio\[VS version dir]\ProjectAssemblies.
- Reset Visual Studio settings (see here for more details). For example, in Visual Studio 2019,
  1. Open cmd.
  2. Switch to C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\Common7\IDE folder.
  3. Run `devenv/resetsettings`.

> ⚠ The Visual Studio settings reset may result in the loss of important data.

Timeout error appears on running Angular(Core) samples for Web Designer and JSViewer with default settings

Symptoms: The timeout error sometimes appears when running the WebDesigner_Angular(Core) and JSViewer_Angular(Core) samples with default settings. See Web Samples for more information.

Cause: The connection timeout period is not sufficient and must be increased.

Solution: To increase the solution timeout period, add the following code to the Startup.cs file.

```
if (env.IsDevelopment())
 {
 spa.UseAngularCliServer(npmScript: "start");
 spa.Options.StartupTimeout = TimeSpan.FromSeconds(200); //timeout
 }
```

Section Report Troubleshooting

Blank pages printed between pages, or a red line appears in the viewer

Symptoms: Blank pages are printed between pages of the report.

Cause: This problem occurs when the PrintWidth plus the left and right margins exceeds the paper width. For example, if the paper size were set to A4, the PrintWidth plus the left and right margins cannot exceed 8.27"; otherwise blank pages will be printed. At run time, ActiveReports marks a page overflow by displaying a red line in the viewer at the position in which the breach has occurred.

Solution: Adjust the PrintWidth in the report designer using either the property grid or by dragging the right edge of the report. Adjust page margins, height, and width either through the print properties dialog box (in the Report menu under Settings), or programmatically in the Report_Start event.

Copying reports results in stacked controls

Symptoms: A report file copied into a new project has all of its controls piled up at location 0, 0.

Cause: The report has become disconnected from its resource file. When you set a report"s Localizable property to True, the Size and Location properties of the report"s controls are moved to the associated *.resx file, so if you copy or move the report, you must move the *.resx file along with it.

Solution: When you copy a report"s *.vb or *.cs file from one project's App_Code folder into the App_Code folder of a new project, you need to also copy its *.resx file from the original project"s App_GlobalResources folder into the new project's App_GlobalResources folder.

No data appears in a report containing the OleObject control

Symptoms: No data appears in a report containing the OleObject control.

Cause: This issue occurs when the Microsoft .NET Framework 4.6.2 or above is used and the useLegacyV2RuntimeActivationPolicy attribute is not set to True.

Solution: Open the app.config file and set the **useLegacyV2RuntimeActivationPolicy** attribute to true.

XML code. Paste INSIDE the app.config file.

```
<configuration>
<startup useLegacyV2RuntimeActivationPolicy="true">
<supportedRuntime version="MyRunTimeVersion"/>
</startup>
</configuration>
```

An error message appears in the Fields list

Symptoms: An error message is displayed in the Fields list in the Report Explorer instead of the fields.

Cause: This is an expected error if no default value is given for a parameter. If the field is a data type other than text, memo, or date/time in Access, the report still runs normally.

Solution: To display the fields in the Fields list in the Report Explorer, supply a default value for the parameter in the Properties Window, or in the SQL query as below:

SQL Query

```
<%Name | PromptString | DefaultValue | DataType | PromptUser%>
```

Only the **Name** parameter is required. To use some, but not all, of the optional parameters, use all of the separator characters but with no text between one and the next for unused parameters. For example:

SQL Query

```
<%Name | | DefaultValue | |%>
```

An unhandled exception of type "System.Data..." occurs when the report is run

Symptoms: When the report is run, an exception like the following occurs: "An unhandled exception of type "System.Data.OleDb.OleDbException" occurred in system.data.dll"

Cause: If the field is a text, memo, or date/time data type in Access, the parameter syntax requires single quotes for text or memo fields, or pound signs for date/time fields. Please note that for different data sources, these requirements may differ.

Solution: To avoid the exception when the report is run against an Access database, use pound signs for date/time values, or single quotes for string values in your SQL query, for example:

SQL Query

```
#<%InvoiceDate | Choose invoice date: | 11/2/04 | D | True%>#
```

or

SQL Query

```
"<%Country | Country: | Germany | S | True%>"
```

User is prompted for parameters for subreports even though they are supplied by the main report

Symptoms: The parameter user interface pops up at run time asking for a value even though the main report is supplying the parameter values for the subreports.

Cause: The default value of the ShowParameterUI property of the report is True.

Solution: Set the ShowParameterUI property of the report to False. This can be done in the property grid or in code in the ReportStart event.

The viewer shows the report on the wrong paper size

Symptoms: In the viewer, the report renders to a different paper size than the one specified.

Cause: ActiveReports polls the printer driver assigned to the report to check for clipping, margins, and paper sizes supported by the printer. If the paper size specified for the report is not supported by the printer, ActiveReports uses the printer's default paper size to render the report.

Solution: If the report is to be printed, the printer assigned to the report must support the paper size and margins. Please note that any changes to the print settings in code must be made in or before the **ReportStart** event. To use custom paper sizes not supported by the driver, set the **PrinterName** to an empty string to use the ActiveReports virtual print driver. This does not allow printing, but is recommended for reports that are only exported or viewed. This prevents ActiveReports from making a call to the default printer driver. Use the following code in the **ReportStart** event, or just before .Run is called.

C# code. Paste INSIDE the ReportStart event.

```
this.Document.Printer.PrinterName = '';
```

Visual Basic.NET code. Paste INSIDE the ReportStart event.

```
Me.Document.Printer.PrinterName = ''
```

The PaperHeight and PaperWidth properties, which take a float value defined in inches, have no effect unless you set the PaperKind property to Custom. Here is some sample code which can be placed in the ReportStart event, or just before .Run.

C# code. Paste INSIDE the ReportStart event.

```
this.PageSettings.PaperKind = Drawing.Printing.PaperKind.Custom;
this.PageSettings.PaperHeight = 2;
//sets the height to two inches
this.PageSettings.PaperWidth = 4;
//sets the width to four inches
```

Visual Basic.NET code. Paste INSIDE the ReportStart event.

```
Me.PageSettings.PaperKind = Drawing.Printing.PaperKind.Custom
Me.PageSettings.PaperHeight = 2
'sets the height to two inches
Me.PageSettings.PaperWidth = 4
'sets the width to four inches
```

Custom paper sizes do not work

Symptoms: Custom paper sizes do not work.

Cause: You can create more than one custom paper size, so setting only the **PaperKind** property is not enough to create a custom paper size.

Solution: In addition to setting the **PaperKind** property to **Custom**, you must also set the **PaperName** property to a unique string.

An exception relating to System.Data.SqlClient occurs on previewing section reports in Windows Forms Viewer or WPF Viewer in the .NET Core 3.1 desktop application

Symptoms: The "Could not load file or assembly 'System.Data.SqlClient" exception occurs when you preview a section report in the Windows Forms Viewer or WPF Viewer in the .NET Core 3.1 desktop application. This exception also occurs when you design a section report in the Designer in the .NET Core 3.1 desktop application.

Cause: This is a Microsoft compatibility issue.

Solution: You need to manually add the Microsoft.Windows.Compatibility NuGet package. For more information, see this article.

System.NotSupportedException occurs on previewing section reports with scripts in Windows Forms Viewer or WPF Viewer in the .NET Core desktop applications

Symptoms: When you preview section reports with scripts using Windows Forms Viewer, WPF Viewer, and Windows Designer components in .NET Core applications, the

exception "System.NotSupportedException. No data is available for encoding 1252" occurs. For information on defining a custom encoding, see the documentation for the Encoding.RegisterProvider method.

Cause: You need to register encodings before using .NET Core applications with Windows Forms Viewer, WPF Viewer, and Windows Designer components.

Solution: To avoid this situation, please do the following.

1. Add the **System.Text.Encoding.CodePages.dll** assembly from the NuGet package.
2. Add **Encoding.RegisterProvider** to Program.cs.

```
static void Main()
{
System.Text.Encoding.RegisterProvider(System.Text.CodePagesEncodingProvider.Instance);Application.EnableVisualStyles();
Application.SetCompatibleTextRenderingDefault(false);
Application.Run(new DesignerForm());
}
```

System.MissingMethodException occurs when running the Windows Forms Viewer on the .NET Core 3.1

Symptoms: When running the Windows Forms Viewer on the .NET Core 3.1 Runtime to preview a section report, the "System.MissingMethodException" exception occurs.

Cause: This is a Microsoft compatibility issue.

Solution: You should change the PowerToolsLicenseProvider.RetrieveLicenseFromAppDomainCache() method to avoid using the AppDomain.CurrentDomain.SetupInformation.LicenseFile property while running the Windows Forms Viewer on the .NET Core 3.1.

An exception "Could not load type 'System.Data.OleDb.OleDbFactory' from assembly 'System.Data'" occurs when running the Windows Forms Viewer on the .NET Core 3.1 Runtime

Symptoms: When running the Windows Forms Viewer on the .NET Core 3.1 Runtime to preview a section report, the "Could not load type 'System.Data.OleDb.OleDbFactory' from assembly 'System.Data'" exception occurs.

Cause: This is a Microsoft compatibility issue.

Solution: You must disable the use of OleDbFactory in the DataExtensionsConfigurationHandler.GetDefaultDataProviders() method.

An exception "System.IO.FileNotFoundException' object" occurs when running a section report in an ASP.NET Core 3.1 project

Symptoms: When running a section report in an ASP.NET Core 3.1 project, the "System.IO.FileNotFoundException' object" exception occurs.

Cause: Due to the Windows Forms dependancy of section reports, you need to add more references to the project.

Solution: To avoid this situation, please do the following.

1. Install the Microsoft.Windows.Compatibility NuGet Package.
2. Add the following code to the project csproj file.

```
<ItemGroup>
<FrameworkReference Include="Microsoft.WindowsDesktop.App" />
</ItemGroup>
```

3. Add the following code to Startup.cs.

Startup.cs

```
public void ConfigureServices(IServiceCollection services)
{Encoding.RegisterProvider(CodePagesEncodingProvider.Instance); }
```

4. Build and run the solution.

Page/RDL Report Troubleshooting

An expression containing a numeric field name does not display any data at run time.

Symptoms: An expression containing a numeric field name does not display any data at runtime.

Cause: Visual Basic syntax does not allow an identifier that begins with a number.

```
i.e. =Fields!2004.Value
```

Solution: Make the numeric field name a string.

```
i.e. =Fields("2004").Value or, =Fields.Item("2004").Value
```

DataSet field in PageHeader of an RDL report

Symptoms: Cannot set a dataset field (bound field) in the PageHeader of an RDL report.

Cause: ActiveReports is based on the RDL 2005 specifications, therefore, referencing datasets in the PageHeader of an RDL report is not supported.

Solution: There is no direct way to add a DataField in a PageHeader, however, as a workaround you can create a hidden report parameter that is bound to your dataset and has the default value set to your expression. For example, `="*" & First(Fields!name.Value)`. You can then use this parameter in the page header. Alternatively, you can use a Page report, which lets you place data fields anywhere on a page.

Exception thrown when using Viewer.Document property

Symptoms: An exception is raised when **Viewer.Document ('Document Property' in the on-line documentation)** is used with a page report or RDL report.

Cause: Document property is available for section reports only.

Cannot add assembly reference created in .NET Framework 4.0 or above in Page Reports/RDL Reports

Symptoms: Cannot add assembly reference created in .NET Framework 4.0 or above in Page Reports/RDL Reports of the stand-alone designer application.

Cause: The Stand-alone Designer application was created using the .NET 3.5 framework, therefore it cannot load .NET 4.0 assemblies.

WPF Viewer Troubleshooting

Toolbox for WPF Viewer is missing from WPF Project

Symptoms: The Toolbox for WPF Viewer does not appear in a new WPF project in Visual Studio on adding WPF package, GrapeCity.ActiveReports.Viewer.Wpf.

Cause: It is a Visual Studio limitation where XAML Hot Reload does not work correctly.

Solution: The XAML Hot Reload should be enabled. To enable the XAML Hot Reload, follow these steps.

1. Go to **Debug** > **Options** > **General**.
2. Select options **Enable UI Debugging Tools for XAML** and **Enable XAML Hot Reload**.
3. Open MainWindow.xaml. The Toolbar should now display the **WPF Viewer** tab.

See Troubleshooting XAML Hot Reload for more information.

TargetInvocationException occurs when running the WPF browser application

Symptoms: When running the WPF browser application, the TargetInvocationException occurs.

Cause: The WPF browser application does not support Partial Trust.

Solution: Make sure that the WPF browser application uses Full Trust. To do that, in the Visual Studio Project menu, go to **YourProject Properties** and on the **Security** tab, under **EnableClickOnce** security settings, select the option **This is a full trust application**.

Design-time error appears on adding the WPF Viewer to the xaml page if the project is targeting .NET Core 3.1.

Symptoms: A design-time error appears on adding the WPF Viewer to the xaml page if the project is targeting .NET Core 3.1.

Cause: This is the .NET limitation.

Solution: To resolve this problem, add the **Microsoft.Windows.Compatibility** NuGet package to the project and then rebuild the project.

Memory Troubleshooting

> **Note**: According to Microsoft it is not necessary to call GC.Collect and it should be avoided. However, if calling GC.Collect reduces the memory leak, then this indicates that it is not a leak after all. A leak in managed code is caused by holding a reference to an object indefinitely. If ActiveReports is holding a reference to an object, then the object cannot be collected by the garbage collector.

Symptoms: ActiveReports is consuming too much memory; CPU usage always goes to 100% when using ActiveReports.

Cause: There are several reasons why too much memory may be consumed:

The report is not being disposed of properly

Cause: The report is not being disposed of properly. The *in*correct syntax is as follows.

C# code.

```
//Incorrect!
rpt.Dispose();
rpt=null;
```

Visual Basic code.

```
'Incorrect!
rpt.Dispose()
rpt=Nothing
```

Solution: The correct syntax for disposing of a section report is as follows.

C# code.

```
//Correct!
rpt.Document.Dispose();
rpt.Dispose();
rpt=null;
```

Visual Basic code.

```
'Correct!
rpt.Document.Dispose()
rpt.Dispose()
rpt=Nothing
```

Machine.Config MemoryLimit setting is insufficient

Cause: Large reports in an ASP.NET application can easily use up the 60% of memory allocated to the ASP.NET worker process by default, which produces an error. In Machine.Config, MemoryLimit specifies the maximum allowed memory size, as a percentage of total system memory, that the worker process can consume before ASP.NET launches a new process and reassigns existing requests.

Solution: Set the **CacheToDisk** property of the document to **True**.

This caches the report to disk instead of holding it in memory. This setting is also detected by the PDF Export, which follows suit, but any other exports still consume memory. Although it is not advised, the ASP.NET worker process memory allocation can also be changed in your Machine.Config file, which is located in a path like: C:\WINDOWS\Microsoft.NET\Framework\v4.0.30319\Config\. Search the Machine.Config file for memoryLimit, which is located in the processModel.

Report never finishes processing

Cause: In some cases, very large reports can consume so much memory that the report never finishes processing. Some of the things that can cause this include:

1. Many non-repeating images, or a high resolution repeating image
2. Instantiating a new instance of a subreport each time the format event of a section fires
3. Using a lot of subreports instead of grouping with joins in the SQL query
4. Pulling in all of the data when only a few fields are needed (e.g. **Select * from db** instead of **Select First, Last, Address from db**)

Solution: In cases where the report is too large to run any other way, the **CacheToDisk** property may be set to **True**. This property should only be used when there is no other way to run the report to completion. Before resorting to this method, please see the Optimizing Section Reports topic.

Task manager indicates the current "working set" of the process

Cause: If inflated memory usage is seen in the Task Manager it is not necessarily in use by the code. Task manager indicates the current "working set" of the process and, upon request, other processes can gain access to that memory. It is managed by the Operating System.

Solution: For an example of some working set behavior anomalies (which are considered normal), create a WinForms application and run it. Look in Task Manager at the working set for that process (it should be several megabytes), then minimize and maximize the form and notice that the working set reclaims to <1MB. Obviously, the code was not using all that memory even though Task Manager showed that it was allocated to that process. Similarly, you'll see ASP.NET and other managed service processes continue to gradually grow their working set even though the managed code in that process is not using all of it. To see whether this is the case, try using the two lines of code below in a button Click event after running the project.

```
System.Diagnostics.Process pc = System.Diagnostics.Process.GetCurrentProcess();
pc.MaxWorkingSet = pc.MinWorkingSet;
```

If that reclaims the memory then the Operating System trimmed the working set down to the minimum amount necessary and this indicates that the extra memory was not actually in use.

Web Applications Troubleshooting

PDF opens in a new window when an application contains the WebViewer

Symptoms: When using Internet Explorer and Acrobat Reader to view a page containing a WebViewer in PDF mode, the resulting PDF always opens in a new window.

Cause: Acrobat Reader is only available in a 32-bit version. When the 64-bit version of Internet Explorer is used, it opens up an instance of the 32-bit version of Internet Explorer so that the plug-in and the PDF can load, rendering the resulting PDF in a new window.

Solution:

- Install a PDF reader plug-in that is 64-bit compatible.
  OR
- Use the 32-bit version of Internet Explorer.

The report in the HTML viewer type does not look exactly like the other viewer types

Symptoms: The report in the HTML viewer type does not look exactly like the other viewer types.

Cause: The HTML format is not WYSIWYG. It does not support the following items:

- Line control
- Control borders
- Shapes (other than filled rects)
- CrossSectionBox and CrossSectionLine controls
- Overlapping controls

Solution: Try to avoid using the above items in reports which are shown in HTML format.

Blank reports with the AcrobatReader viewer type on the production web server

Symptoms: In the WebViewer, reports render correctly with the HTML ViewerType but they show up blank with the AcrobatReader ViewerType on the production web server.

Cause: .ArCacheItem is not set up in your IIS extension mappings.

Solution:

1. From the Start menu, choose **Control Panel**, then **Administrative Tools**, then **Internet Information Services**.
2. Right-click your Default Web Site and choose **Properties**.
3. On the Home Directory tab, click the **Configuration** button.
4. On the Mapping tab, check the Extension column to see whether .ArCacheItem appears. If not, click **Add**.
5. In the Add/Edit Application Extension Mapping dialog that appears, click **Browse** and navigate to (Windows)\Microsoft.NET\Framework\vx.0.
6. In the Open dialog, change **Files of type** to Dynamic Link libraries (*.dll).
7. Select **aspnet_isapi.dll** and click **Open**.
8. In the Extension textbox type **.ArCacheItem**.
9. Click the **Limit to** radio button and type **GET,HEAD,POST,DEBUG**.
10. Ensure that the **Script engine** check box is selected and the **Check that file exists** check box is cleared.
11. Click **OK**.

A web page gets refreshed on performing any action in the WebViewer

Symptoms: A Web browser gets reloaded on performing any action in the WebViewer.
Cause: This is the default behavior of WebForms with the WebViewer.
Solution: Make sure that the WebViewer control is placed outside the Form tag with a runat=server attribute.

PlatformNotSupportedException occurs on using WebViewer, JSViewer, and Web Designer

Symptoms: PlatformNotSupportedException occurs in Web Applications using WebViewer, JSViewer, and Web Designer in Classic Pipeline Mode.
Cause: The WebViewer, JSViewer, and Web Designer are supported only in the Integrated pipeline mode.
Solution: Change the Application Pool mode as follows:

1. Open **IIS Manager**.
2. Go to **Application Pools**.
3. Select the application pool where your app runs.
4. Select **Basic Settings.**
5. In the **Edit Application Pool** dialog, change the **Managed Pipeline Mode** to **Integrated**.

The "Report not found" error occurs when a report name contains special symbols

Symptoms: When you specify a report name with special symbols, e.g. webViewer.ReportName="Folder\Report.rdlx", you may get a "Report not found" error.

Cause: Additional code needs to be added to the Web.config file to have the WebViewer use report names with the corresponding folders.

Solution: Add the following code to the Web.config file.

Paste inside the Web.config file

```
<system.web>
<httpRuntime requestPathInvalidCharacters="" requestValidationMode="2.0"/>
<pages validateRequest="false"/>
</system.web>

<system.webServer>
<security>
<requestFiltering allowDoubleEscaping="true"/>
</security>
</system.webServer>
```

The 'The type or namespace name 'Linq' does not exist in the namespace 'System'' error occurs when adding the WebViewer control in an ASP.NET Web Site project

Symptoms: When you add the WebViewer control in an ASP.NET Web Site project, the 'The type or namespace name 'Linq' does not exist in the namespace 'System' error occurs.

Cause: This is a known NuGet limitation.

Solution: You should install or upgrade the Microsoft.CodeDom.Providers.DotNetCompilerPlatform NuGet package.

"This application will be terminated because it was built without a license for PageReport" error occurs on deploying an application on Azure Functions Application.

Symptoms: When you deploy an application in an Azure Functions application, error "This application will be terminated because it was built without a license for PageReport" occurs.

Cause: This is because the application being deployed is not licensed properly.

Solution: You should follow the steps provided in the Licensing Compiled Code topic for correctly licensing the application before deploying it to the Azure Functions application.