

Table of Contents

.	Table of Contents	0-12
.	ActiveReports Developer 7	13
.	ActiveReports Developer Guide	13
.	Welcome to ActiveReports Developer 7	13-14
.	What's New	14-17
.	ActiveReports Editions	17-23
.	Installation	23-24
.	Requirements	24
.	Install ActiveReports Developer	24-25
.	Installed Files	25-28
.	Installing Help in Visual Studio 2010	28-29
.	Side-by-Side Installation	29-30
.	ComponentOne Copyright Notice	30
.	End User License Agreement	30-31
.	.NET Framework Client and Full Profile Versions	31
.	Redistributable Files	31-32
.	License Your ActiveReports	32-38
.	Upgrading Reports	38-39
.	Breaking Changes	39-42
.	Converting Crystal Reports/MS Access Reports	42-45
.	Getting Started	45
.	Adding ActiveReports Controls	45-46
.	Adding an ActiveReport to a Project	46-47
.	Adding a Data Source to a Report	47
.	Viewing Reports	47-48
.	Using the Viewer	48-55
.	ActiveReports and the Web	55
.	Getting Started with the Web Viewer	55-57

.	Using the Flash Viewer	57-60
.	Using the HTML Viewer	60-62
.	Working with HTML Viewer using Javascript	62-63
.	Medium Trust Support	63-64
.	Using the Silverlight Viewer	64-67
.	Using the WPF Viewer	67-72
.	Adding an ActiveReports Application	72-73
.	Concepts	73-74
.	ActiveReports Designer	74-76
.	Design View	76-78
.	Report Menu	78-79
.	Designer Tabs	79-81
.	Designer Buttons	81-85
.	Page Tabs	85-86
.	Toolbar	86-90
.	Report Explorer	90-91
.	Exploring Section Reports	91-92
.	Exploring Page Reports	92-95
.	Toolbox	95
.	Properties Window	95
.	Rulers	95-96
.	Scroll Bars	96-97
.	Snap Lines	97-98
.	Zoom Support	98-99
.	Report Types	99-102
.	CPL Page Report	102-104
.	FPL Page Report	104-105
.	Code-Based Section Report	105-106
.	XML-Based Section Report	106
.	Page Report Concepts	106-107

.	Page Report Toolbox	107-108
.	BandedList	108-112
.	Barcode	112-117
.	Bullet	117-120
.	Calendar	120-123
.	Chart	123-131
.	Chart Data Dialog	131-138
.	CheckBox	138-140
.	Container	140-142
.	FormattedText	142-144
.	Image	144-146
.	Line	146-147
.	List	147-150
.	Matrix	150-155
.	OverflowPlaceHolder	155-157
.	Shape	157-158
.	Sparkline	158-162
.	Subreport	162-164
.	Table	164-169
.	TextBox	169-173
.	Data Sources and Datasets	173
.	Report Data Source Dialog	173-177
.	DataSet Dialog	177-179
.	Shared Data Source (RDSX)	179-180
.	Expressions	180-182
.	Common Values	182-183
.	Common Functions	183-187
.	Using Script in a Page Report	187-189
.	Report Dialog	189-191
.	FixedPage Dialog	191-194

.	Grouping Data (Page Layout)	194-196
.	Add Page Numbering	196-197
.	Themes	197-198
.	Rendering	198-200
.	Master Reports	200-202
.	Data Visualizers	202
.	Icon Set	202-206
.	Range Bar	206-209
.	Range Bar Progress	209-212
.	Data Bar	212-215
.	Color Scale 2	215-218
.	Color Scale 3	218-221
.	Custom Resource Locator	221-225
.	Section Report Concepts	225
.	Section Report Toolbox	225-226
.	Label	226-229
.	TextBox (Section Report)	229-232
.	CheckBox (Section Report)	232-233
.	RichTextBox	234-236
.	Shape (Section Report)	236-237
.	Picture	237-238
.	Line (Section Report)	238-239
.	PageBreak	239-240
.	Barcode (Section Report)	240-246
.	SubReport (Section Report)	246-247
.	OleObject	247-248
.	ChartControl	248-249
.	ReportInfo	249-251
.	CrossSection Controls	251-253

.	Section Report Structure	253-255
.	Section Report Events	255-259
.	Scripting in Section Reports	259-260
.	Report Settings Dialog	260-261
.	Grouping Data in Section Reports	261-264
.	Date, Time, and Number Formatting	264-265
.	Optimizing Section Reports	265-267
.	CacheToDisk and Resource Storage	267
.	Text Justification	267-268
.	Multiline in Report Controls	268
.	Line Spacing and Character Spacing	268-269
.	Exporting	269-270
.	Export Filters	270
.	HTML Export	270-272
.	PDF Export	272-274
.	Text Export	274-275
.	RTF Export	275-276
.	Excel Export	276-277
.	TIFF Export	277-278
.	Font Linking	278-279
.	Interactive Features	279
.	Parameters	279-283
.	Filtering	283-284
.	Drill-Down Reports	284
.	Linking in Reports	284-285
.	Document Map	285-286
.	Sorting	286-288
.	Annotations	288-289
.	Windows Forms Viewer Customization	289-290
.	Designer Control (Pro Edition)	290-291

.	Shrink Text to Fit in a Control	291
.	Standalone Designer and Viewer	291-293
.	Localization	293
.	Cultures	293-299
.	How To	299
.	Page Report How To	299-301
.	Work with Data	301
.	Connect to a Data Source	301-302
.	Add a Dataset	302-304
.	Create and Edit a Shared Data Source	304
.	Bind a Page Report to a Data Source at Run Time	304-313
.	Work with Report Controls and Data Regions	313
.	Grouping in a FixedPage	313-314
.	Grouping in a Data Region	314-321
.	Set Detail Grouping In Sparklines	321-322
.	Set Filters in Page Reports	322-325
.	Create Common Page Reports	325
.	Create Top N Report	325-326
.	Create Red Negatives Report	326-327
.	Create Green Bar Report	327
.	Create a Bullet Graph	327-328
.	Create a Whisker Sparkline	328-329
.	Add Parameters in a Page Report	329-330
.	Create an ALL Parameter	330-331
.	Add a Cascading Parameter	331-333
.	Set a Hidden Parameter	333-334
.	Create and Add Themes	334-335
.	Customize and Apply a Theme	335-336
.	Use Constant Expressions in a Theme	336
.	Set Up Collation	336-337

.	Add Hyperlinks	337
.	Add Bookmarks	337-339
.	Create and Use a Master Report	339-340
.	Export a Page Report (Export Filter)	340-342
.	Export a Page Report (Rendering Extension)	342-343
.	Sort Data	343-346
.	Allow Users to Sort Data in the Viewer	347-348
.	Create a Drill-Down Report	348-349
.	Set a Drill-Through Link	349-351
.	Add Items to the Document Map	351-353
.	Change Page Size	353-354
.	Add Page Breaks in CPL	354
.	Add Totals and Subtotals in a Data Region	354-359
.	Add Static Rows and Columns to a Matrix	359-360
.	Cell Merging In Matrix	360-361
.	Section Report How To	362
.	Work with Data in Section Reports	362-363
.	Bind Reports to a Data Source	363-368
.	Add Grouping in Section Reports	368-369
.	Modify Data Sources at Run Time	369-371
.	Work with Report Controls	371
.	Add Field Expressions	371-373
.	Display Page Numbers and Report Dates	373-374
.	Load a File into a RichTextBox Control	374-378
.	Use Custom Controls on Reports	378-379
.	Create Common Section Reports	379-380
.	Create Top N Reports	380-381
.	Create a Summary Report	381-383
.	Create Green Bar Reports	383-384

.	Inherit a Report Template	384-385
.	Change Ruler Measurements	385-386
.	Print Multiple Copies, Duplex and Landscape	386-388
.	Conditionally Show or Hide Details	388-389
.	Add Parameters in a Section Report	389-391
.	Add and Save Annotations	391-393
.	Add Bookmarks	393-395
.	Add Hyperlinks	395-398
.	Use External Style Sheets	398-399
.	Insert or Add Pages	399-402
.	Embed Subreports	402-403
.	Add Code to Layouts Using Script	403-408
.	Export a Section Report	408-410
.	Save and Load RDF Report Files	410-411
.	Save and Load RPX Report Files	411-413
.	Customize, Localize, and Deploy	413-414
.	Localize Reports, TextBoxes, and Chart Controls	414-415
.	Localize ActiveReports Resources	415-416
.	Customize the Viewer Control	416-418
.	Localize the Viewer Control	418-420
.	Deploy Windows Applications	420-421
.	Deploy Web Applications	421-422
.	Localize the End User Report Designer	422-423
.	Customize the FlashViewer Toolbar	423-426
.	Localize the Flash Viewer	426-427
.	Configure HTTPHandlers in IIS 6.x	427-429
.	Configure HTTPHandlers in IIS 7.x	429-432
.	Use Fields in Reports	432-434
.	Use Advanced Printing Options	434-435
.	Provide One-Touch Printing in the WebViewer (Pro Edition)	435

·	Provide PDF Printing in the Silverlight Viewer (Pro Edition)	435-437
·	Print Methods In ActiveReports Developer	437-439
·	Samples and Walkthroughs	439
·	Samples	439-440
·	Page Report	440
·	FPL Samples	440
·	API	440
·	Custom Resource Locator	440-441
·	Data	441
·	Page Unbound Data Sample	441-442
·	Layout	442
·	FPL Report Loader Sample	442-444
·	CPL Samples	444-445
·	API	445
·	Create Report	445-446
·	DataSet DataSource	446-447
·	Normalized DataSet	447-448
·	OleDb DataSource	448-449
·	Report Wizard	449-450
·	Xml Data Provider	450-451
·	Layouts	451
·	CPL Report Loader	451-455
·	Web	455
·	Page Reports On Web	455-456
·	Section Report	456
·	Data	456-457
·	Bound Data Sample	457-458
·	IList Binding Sample	458-460
·	LINQ Sample	460
·	Unbound Data Sample	460-462

.	XML Sample	462-463
.	Layout	463
.	Annual Report Sample	464-465
.	Category Selection Sample	465-466
.	Charting Sample	466-467
.	Cross Section Controls Sample	467-469
.	Cross Tab Report Sample	469-470
.	Inheritance Sample	470-471
.	Layout Loader Sample	471-475
.	Style Sheets Sample	475-476
.	Subreport Sample	476-479
.	Preview	479
.	Custom Annotation Sample	479-480
.	Custom Preview Sample	480-484
.	Hyperlinks and DrillThrough Sample	484-486
.	Print Multiple Pages per Sheet Sample	486-487
.	RDF Viewer Sample	487-488
.	Summary	488
.	Calculated Fields Sample	488-489
.	Data Field Expressions Sample	489-490
.	Web	490
.	Standard Edition Web Sample	490-492
.	Professional	492
.	Custom Data Provider Sample	492-494
.	Digital Signature Sample	494-495
.	End User Designer Sample	495-497
.	Professional Web Sample	497-503
.	Silverlight Viewer Sample	503-504
.	ActiveReports 7 with MVC Sample	504-506

.	Walkthroughs	506-507
.	Page Report Walkthroughs	507-508
.	Single Layout Reports	508-511
.	Overflow Data in Multiple Pages	512-517
.	Overflow Data in a Single Page	517-520
.	Collate Multiple Copies of a Report	520-523
.	Subreport in a CPL Report	523-531
.	Columnar Layout Reports (CPL)	531-534
.	BandedList Reports	534-538
.	Matrix Reports	539-543
.	Reports with XML Data	543-547
.	Master Detail Reports	547-550
.	Expressions in Reports	550-552
.	Recursive Hierarchy Reports	552-556
.	Reports with Parameterized Queries	556-560
.	Reports with Custom Code	560-564
.	Reports with Stored Procedures	564-566
.	Charts (Page Report)	566-570
.	Interactive Reports	570
.	Reports with Bookmarks	570-575
.	Drilldown Reports	575-576
.	Drill-Through Reports	576-583
.	Parameterized Reports	583-586
.	Custom Web Exporting (Page Report)	586-591
.	Custom Data Provider	591-626
.	Section Report Walkthroughs	626
.	Basic Data Bound Reports	626-628
.	Subreport Walkthroughs	628-629
.	Subreports with Run-Time Data Sources	629-633
.	Subreports with XML Data	633-637

.	Chart Walkthroughs	637
.	Bar Chart	637-639
.	3D Pie Chart	639-641
.	Financial Chart	641-644
.	Unbound Chart	644-646
.	Custom Web Exporting (Std Edition)	646-651
.	Custom HTML Outputter (Std Edition)	651-657
.	Basic XML-Based Reports (RPX)	657-661
.	Layout Files with Embedded Script	661
.	Script for Simple Reports	661-668
.	Script for Subreports	668-676
.	Address Labels	676-678
.	Columnar Reports	678-681
.	Overlaying Reports (Letterhead)	681-686
.	Group On Unbound Fields	686-693
.	Mail Merge with RichText	693-700
.	Run Time or Ad Hoc Reporting	700
.	Run Time Layouts	700-710
.	Run Time Data Sources	710-714
.	Creating a Basic End User Report Designer (Pro Edition)	714-718
.	Customizing the Flash Viewer UI	718-724
.	Customizing the HTML Viewer	724-727
.	Web Services (Std Edition)	727
.	DataSet Windows Application	727-729
.	Document Web Service	729-731
.	Document Windows Application	731-733
.	DataSet Web Service	733-734
.	Basic Spreadsheet with SpreadBuilder	735-737
.	WPF Viewer	737-740
.	Troubleshooting	740-750

. . . Class Library	750-751
. . Index	752-760

ActiveReports Developer 7

This is the help file for ActiveReports, reporting software for use in Visual Studio 2008, 2010 or 2012.

In This Documentation

ActiveReports Developer Guide

The Developer Guide has many getting started topics and how-to topics with code samples to copy and paste.

Class Library

This is the API documentation with topics for all of the public members of each assembly included with ActiveReports.

ActiveReports Developer 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.

In This Documentation

Welcome to ActiveReports Developer 7

This guide provides basic information on installing and using the product, as well as support, licensing, and what's new.

License Your ActiveReports

This topic walks you through how to license your machine and how to add licensing to any projects created during your evaluation.

Upgrading Reports

This topic provides information about upgrading reports from ActiveReports 6 and Data Dynamics Reports, and about converting MS Access Reports and Crystal Reports with the Import Wizard.

Getting Started

This section provides an overview of the interface and where to find everything you need to get started designing reports.

Concepts

This section provides information on what you can do with ActiveReports.

How To

This section provides step-by-step instructions for many features.

Samples and Walkthroughs

This section provides a description of the samples available with ActiveReports and step-by-step walkthroughs explaining key features.

Troubleshooting

This section provides troubleshooting symptoms, causes, and solutions to commonly encountered issues.

Welcome to ActiveReports Developer 7

Explore ActiveReports Developer 7.

This section contains information about

What's New

Learn about the new features in ActiveReport Developer.

ActiveReports Editions

Find out which features can be used with Standard and Professional Edition licenses.

Installation

View requirements for installation of ActiveReports, learn what files are installed and how to verify your installation, and find installation troubleshooting tips.

ComponentOne Copyright Notice

Explains ComponentOne copyright information.

End User License Agreement

Understand the terms of the ActiveReports License Agreement and Limited Warranty.

.NET Framework Client and Full Profile Versions

Provides details of assemblies that are compliant with .NET Framework 3.5 Client Profile and .NET Framework 4.0 Client Profile

Redistributable Files

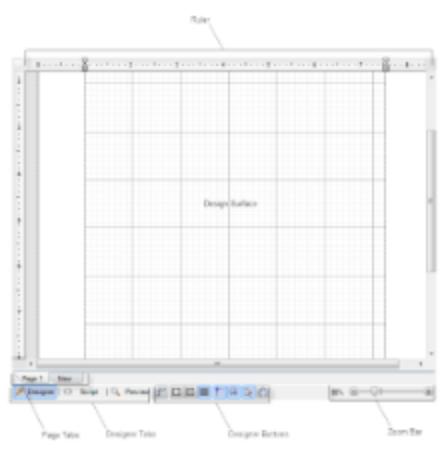
Find out the list of files that may be distributed.

What's New

ActiveReports Developer contains many new features along with the popular ActiveReports engine and report viewers, plus the powerful Data Dynamics Reports designer.

New Page Layout

In the new Page layout, you can design reports at the page level without any banded sections. This lets you place controls anywhere on the report. This layout has two variations, Continuous Page Layout (CPL) and Fixed Page Layout (FPL). In a CPL report, controls grow vertically to accommodate data. In an FPL report, controls do not change in size based on the data, but you can use an OverflowPlaceholder to handle any extra data. For more information, see **FPL Page Report** and **CPL Page Report**.



- **Interactive** report options include drill-down links, bookmark links, hyperlinks, document maps, sorting, and filtering. Users can benefit from these features when viewing reports at run-time.
- **Shared data source** refers to a file in RDSX format that contains data connection information. RDSX (Report Data Source XML) is a proprietary file format that functions as a reusable data source for a single report or multiple reports.
- **Data regions** are report controls that can contain other report controls that repeat for every row of data. Report controls that function as data regions are Table, List, BandedList, Matrix and Chart.
- **OverflowPlaceholder** is a rectangular placeholder control. When you use a List, BandedList, Matrix or Table data region on an FPL report, data regions cannot grow, so you can specify an OverflowPlaceholder control to catch the overflow.
- **Theme Editor** allows you to create themes by setting colors, fonts, images, and constant expressions that you save

to an .rdlx-theme file. You can add one or more themes to a report. If a report has multiple themes, you can set up **collation** to control the page order.

- **Expression Editor** allows you to use an expression to set the value of a control, or set conditions under which certain styles apply. You can enter Microsoft Visual Basic® .NET expressions in many properties using the Properties Window or the Expression Editor Dialog.
- **Data Visualizer** has several ways to display data in easy-to-comprehend formats that are small enough to use in line with text in the report. You can open the Data Visualizer dialog within the Image or BackgroundImage properties of the Image and TextBox report controls.

[Learn More](#) | [Report Types](#) | [Shared Data Source](#) | [Page Report Toolbox](#) | [OverflowPlaceHolder](#) | [Themes](#) | [Expressions](#) | [Single Layout Reports](#) | [Overflow Data in a Single Page](#) | [Overflow Data in Multiple Pages](#)

Excel Improvements

- The Excel export filter now supports Excel 2007.
- New page settings features include Orientation and PaperSize.
- New security features include Password, ProtectedBy, ReadOnlyRecommended, and WritePassword.

[Learn More](#) | [Excel export](#)

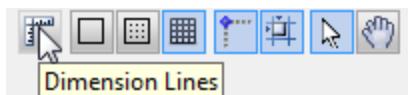
Barcodes

- New barcode types include **DataMatrix**, **Matrix_2_of_5**, and **IntelligentMail**.
- The new **NWRatio** property allows you to modify the ratio between narrow and broad bars for some barcode types.
- New **QuietZone**, **CaptionGrouping**, and **Rotation** properties.
- Enhancements to barcodes include encoding enhancement in **QRCode** and caption separation capability in **EAN/UPC**.

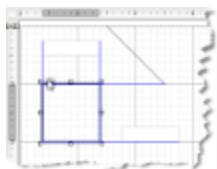
[Learn More](#) | [Barcode](#)

New Designer

Designer buttons located to the right of the designer tabs along the bottom of the designer provide fast access to layout guides.



- **Snap Lines** mode 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 or section, and when you drop it, it snaps into place in perfect alignment.



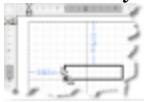
- **Snap to Grid** mode aligns the control you are dragging with grid lines on the report design surface. When you drop the control, it snaps into place in alignment with the nearest grid mark. To place your controls freely on the report design surface, turn this setting off.



- **Excel-like auto scrolling** works in 4 directions, top, bottom, left and right. This option is very useful when you drag a control beyond the visible area of the report design surface. The report scrolls in the direction you drag the mouse until it reaches the edge of the design surface.
- **Pan Mode** has a hand cursor that lets you navigate through your report by clicking the left mouse button and dragging the report to the desired position.



- **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 design 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.



- **Report control dialogs** offer an easy way to set relevant properties on report controls.

[Learn More](#) | [Designer Buttons](#) | [Snap Lines](#)

Windows Forms Viewer

- **Selection** mode allows users to select contents of the report, including partial text selection.
- **Snapshot** mode allows users to select areas of the report to copy and paste as an image into any application that accepts pasted images.
- **Page Setup** dialog allows users to change page scaling, set page margins and add a watermark when printing a report.



- **Parameters** pane allows users to select parameters for reports. It shows up automatically for reports with parameters.

[Learn More](#) | [Use Advanced Printing Options](#)

WebViewer (HTML Type)

- **Web Viewer** is now AJAX-based and no longer requires PostBack. For example, **Go to Previous page** and **Go to Next page** buttons in the HTML viewer work via Web service and no longer require PostBack.
- **Parameters** pane allows users to select parameters for reports. To show or hide the Parameters pane in the sidebar, click the **Toggle Sidebar** button in the viewer toolbar.



[Learn More | Using the HTML Viewer](#)

Silverlight Viewer

The Silverlight viewer also has a **Parameters** pane that allows users to select parameters for reports. To show or hide the Parameters pane in the sidebar, click the **Toggle Sidebar** button in the Toolbar.

[Learn More | Using the Silverlight Viewer](#)

WPF Viewer

The WPF viewer is a custom control that allows to view both section and page report layouts. It contains a toolbar and a sidebar with **Thumbnails**, **Search results**, **Document map** and **Parameters** panes.

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

[Learn More | Using the WPF Viewer](#)

PDF/A Support

ActiveReports PDF export filter and PDF rendering extension (Page Layout) provide the PDF/A support that includes versions such as PdfA1a, PdfA1b, PdfA2a, PdfA2b, or PdfA2u.

[Learn More | Rendering](#)

ActiveReports Editions

ActiveReports Developer 7 is an enhancement of the popular ActiveReports engine and report viewers, plus the powerful Data Dynamics Reports designer. It includes the same power and flexibility of ActiveReports and the same integration with the Visual Studio® .NET 2008 and Visual Studio .NET 2010 Environments, plus adds many features.

Available in two editions, Standard and Professional, ActiveReports Developer 7 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 report types
 - C# and VB.NET support with code-based section reports
 - Script support with XML-based section reports
 - Expression support with page 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

- ReportInfo
- Label
- Line
- PageBreak
- OleObject
- Subreport
- Shape
- Picture
- RichTextBox with HTML tag support
- ChartControl 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

Page Reports

- Table data region
- Matrix data region
- Chart data region
- List data region
- BandedList data region
- Calendar data region
- Sparkline data region
- FormattedText with mail merge capabilities and XHTML + CSS support
- Bullet Graph
- BarCode
- TextBox
- Line
- Container
- Shape
- Image
- Subreport
- Overflow Placeholder

Expressions (page reports only)

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

Interactive Features

- Document map (table of contents)
- Bookmark links, hyperlinks, and drill through links
- Parameters
- Drill-down (page reports only)
- 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 only)

Export Filters

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

Export formats	Section report	Page 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. The PDF export includes the PDF/A support.	✓	✓
Rtf: Export reports to RTF, RichText format that opens in Microsoft Word, and is native to WordPad.	✓	✓
Doc: Export reports to Word, 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, EMF, 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 (Excel 2007).	✓	✓
Xml: Export reports to XML, a format that opens in a Web browser or delivers data to other applications.	✗	✓

Import Filters

- Access® Reports
- Crystal Reports
- ActiveReports (older code-based reports only; XML-based reports open without importing)

Stand-Alone Applications

- A Report Designer application and a Report Viewer application are also included for your convenience. They can be opened from the Start menu, and neither requires Visual Studio.
- The Report Designer application contains all the functionality of the integrated Report Designer.
- The Report Viewer application contains all the functionality of the ReportPreview control.

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

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.

Silverlight Viewer Control

- The Silverlight viewer control allows you to provide in- or out-of-browser report viewing in your Silverlight applications.
- Like our other viewers, the Silverlight viewer control offers customization and localization.

WebViewer Control

- The WebViewer control allows quick viewing of ActiveReports on the web as well as printing capability with the AcrobatReader ViewerType enumeration.
- Flash ViewerType enumeration supports multiple browsers and offers customization and localization options.

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.

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.

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.

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	<p>WebViewer: Use this control to display your reports on the Web. Includes viewer types HTML, PDF, and Flash.</p> <p>Silverlight Viewer: Use this control to display your reports in Silverlight 4 or higher, and for out-of-browser viewing.</p> <p>HTTP Handlers: PDF and HTML (compiled report, RPX file)</p> <p>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.</p>	✗	✓
Windows Forms	<p>Designer: Use this control to create a royalty-free, custom designer that your end users can use to create and modify their own reports.</p> <p>ReportExplorer: Use this control along with the Designer control to provide functionality to your users.</p> <p>ToolBox: Use this control along with the Designer control to provide report controls for your users.</p>	✗	✓
WPF Viewer	<p>Viewer: Use this control to display your section and page reports. The Viewer offers the Thumbnails pane, the Parameters pane, the Document map pane, the Search results pane, and the Annotations toolbar.</p> <p>HtmlExport: Export reports to HTML, DHTML, or MHT formats that open in a Web browser.</p> <p>PdfExport: Export reports to PDF, a portable document format that opens in the Adobe Reader.</p> <p>RtfExport: Export reports to RTF, RichText format that opens in Microsoft Word, and is native to WordPad.</p> <p>WordExport: Export reports to Word HTML, a format that opens in Microsoft Word.</p>	✓	✓
Web and Windows Forms	<p>TextExport: Export reports to TXT, plain text, a format that opens in Notepad or any text editor.</p> <p>This export filter can also export reports to CSV, comma separated values, a format that you can open in Microsoft Excel.</p> <p>ImageExport: Export reports to BMP, EMF, GIF, JPEG, TIFF, or</p>	✓	✓

PNG image format.

Note that you can only export section reports to the TIFF image type.
All other image types are for page reports.

PDF Export Advanced Features	XlsExport: Export reports to formats that open in Microsoft Excel, XLS or XLSX (Excel 2007).	✓	✓
	XmlExport: Export reports to XML, a format that opens in a Web browser or delivers data to other applications.	✓	✓
	Digital signatures	✗	✓
	Time stamp	✗	✓
	EUDC	✗	✓
	Select from Japanese embedded fonts or unembedded fonts	✗ *1	✓
	Bold	✗	✓
	Italic	✓	✓
	Multi Language	✓ *2	✓
	PDF/A Support	✓	✓

Integrated Report Designer

Design Format	Section reports support banded layouts.	✓	✓
	Page reports support fixed page layouts (FPL) and continuous page layouts (CPL).		
	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, 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.		
	You can save and load page 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.		
	The BarCode control supports all of the following styles:	✓	✓
	ANSI 3 of 9	ANSI Extended 3 of 9	Code 2 of 5 2 of 5
	Code 25 Matrix	Code 39	Extended Code 39
	Code 128 B	Code 128 C	Code 128 Auto
Report File Formats	Extended Code 93	MSI	PostNet
	EAN-8	EAN-13	UPC-A
	UPC-E1	RoMail RM4SCC	UCC/EAN-128
	Code 49	Japanese Postal	Pdf417
	RSS-14	RSS-14 Truncated	EAN-128 FNC1
			RSS-14 Stacked

Report Controls	RSS-14 Stacked Omnidirectional	RSS Expanded	RSS Expanded Stacked														
		The Chart control supports all of the following styles:		✓	✓												
		<ul style="list-style-type: none"> Common Charts: Area, Bar2D, Bezier, Doughnut/Pie, Line, Scatter, StackedArea, StackedBar, StackedArea100Pct, and StackedBar100Pct 3D Charts: Area3D, Bar3D, ClusteredBar, Line3D, Doughnut3D/Pie, StackedBar3D, and StackedBar3D100Pct XY Charts: Bubble, BubbleXY, LineXY, and PlotXY Financial Charts: Candle, HiLo, and HiLoOpenClose 															
		Other report controls include:		✓	✓												
		<table border="0"> <tbody> <tr> <td>Label</td> <td>TextBox</td> <td>CheckBox</td> <td>Picture</td> </tr> <tr> <td>Line</td> <td>Shape</td> <td>RichText</td> <td>PageBreak</td> </tr> <tr> <td>SubReport</td> <td>ReportInfo</td> <td>CrossSectionLine</td> <td>CrossSectionBox</td> </tr> </tbody> </table>	Label	TextBox	CheckBox	Picture	Line	Shape	RichText	PageBreak	SubReport	ReportInfo	CrossSectionLine	CrossSectionBox			
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), Oracle data, XML data, and unbound data	✓	✓
Printing	You can control the page size, orientation, and margins, as well as specifying bound (double page spread), collating, duplex printing, and paper feed trays.	✓	✓

*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.

Installation

This section helps you understand the installation process.

In this section:

Requirements

Learn about the hardware and software required to run ActiveReports.

Install ActiveReports Developer

Find out how to install the ActiveReports Developer Setup.

Installed Files

Find out what files are installed with ActiveReports Developer and where to locate them.

Installing Help in Visual Studio 2010

Find out how to integrate Help in Visual Studio 2010.

Side-by-Side Installation

Learn about working with ActiveReports Developer 7 and ActiveReports 6 or Data Dynamics Reports on a single machine.

Requirements

To install and use ActiveReports Developer 7, you need the following hardware and software.

Hardware requirements (minimum)

- **Hard drive space:** 200 MB available
- **CD Drive:** For CD package only

Software requirements

- **Operating System:** Windows® XP, Windows™ Vista, Windows 7, Windows Server 2003, Windows Server 2008, or Windows Server 2008 R2
- **Microsoft® .NET Framework Version:** 3.5 SP1, 4.0 or 4.5
- **.NET Framework Client Profile:** 3.5 or 4
- **Microsoft Visual Studio:** 2008, 2010 or 2012

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

- **For Web deployment:** IIS 5.1, 6.0, 7.0 or 7.5 and ASP.NET (version to match the .NET Framework version)
- **Flash Player:** Adobe Flash Player 11
- **Browser:** Microsoft Internet Explorer 7 or higher, Mozilla Firefox 5 or higher, Google Chrome 17.

 **Note:** Microsoft Silverlight 4 Tools is required for the application development with the ActiveReports Silverlight Viewer.

Install ActiveReports Developer

Follow the steps below to install ComponentOne ActiveReports Developer on your machine.

 **Note:** Your machine setup may require you to be logged in as an Administrator to install new software. If this is the case and you do not have Administrator privileges, consult your system administrator.

1. Insert the ComponentOne ActiveReports Developer CD into your CD-ROM drive.
OR
If you have the ComponentOne ActiveReports Developer setup on your system, double-click the Setup.exe file or right-click the file and select **Install**.
2. In the ComponentOne ActiveReports Developer Setup window that appears, on the Welcome screen, click **Next** to continue with installation.



3. On the End-User License Agreement screen that appears, go through the terms in the License Agreement, select the check-box to accept them and click **Next** to continue with installation.



4. On the Installation Options screen that appears, optionally select ComponentOne ActiveReports Developer Samples to install them with the product and click **Next** to continue with installation.

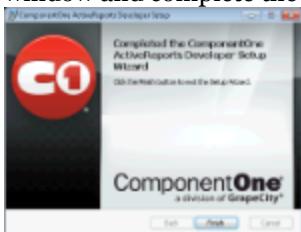


Note: These samples help you in understanding different usage scenarios that the product offers.

5. On the Licensing Options screen that appears, choose out of the three licensing options and click **Install**.
 - Evaluation
 - Activate Now
 - Activate Later



6. Once the installation finishes, a screen notifying the completion of installation appears. Click **Finish** to close the window and complete the installation process.



Installed Files

You can verify your package installation 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 Developer 7.0** appears in the installed products list.

When you install ActiveReports Developer and use all of the default settings, files are installed in the following folders:

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

File (or Folder) Description

ActiveReports Developer (folder)	Shortcut to the folder containing Standalone applications, help files and Samples folder. See the next dropdown for further details.
License Manager	Shortcut to the License Manager application.

C:\ProgramData\Microsoft\Windows\Start Menu\Programs\ComponentOne\ActiveReports Developer

File (or Folder)

Description

ActiveReports Developer Designer	Shortcut to the Standalone Designer application.
ActiveReports Developer Documentation for Microsoft Help Viewer	Shortcut to the integrated help file for Microsoft Help Viewer.
ActiveReports Developer Documentation for Visual Studio .NET 2008	Shortcut to the integrated help file for Visual Studio .NET 2008.
ActiveReports Developer Import	Shortcut to the ActiveReports Developer Import wizard application.
ActiveReports Developer Theme Editor	Shortcut to the ActiveReports Developer Theme Editor application.
ActiveReports Developer Viewer	Shortcut to the Standalone ActiveReports Developer Viewer application.
Samples (folder)	Shortcut to the folder containing sample projects.

C:\Users\YourUserName\Documents\ComponentOne Samples\ActiveReports Developer 7

Folder

Description

Data (folder)	Included sample data files.
Page Reports (folder)	Included Page Report samples.
Section Reports (folder)	Included Section Report samples.

C:\Program Files\ComponentOne\ActiveReports Developer 7 (C:\Program Files (x86)\ComponentOne\ActiveReports Developer 7 on a 64-bit Windows operating system)

File (or Folder)

Description

Deployment (folder)	Includes Flash viewer file, Flash viewer themes, Silverlight localization resources and templates for redistribution.
Help (folder)	Includes integrated Help files, Cabinet files, Help Integration files, Registry objects, Microsoft Help Viewer supporting file and Command files.
Icons (folder)	Includes associated Icons image files.
Localization (folder)	Includes Resource and DOS batch files for localizing ActiveReports Developer components.
Grapecity.ActiveReports.config	XML configuration file.

C:\Program Files\Common Files\ComponentOne\ActiveReports Developer 7 (C:\Program Files (x86)\Common Files\ComponentOne\ActiveReports Developer 7 on a 64-bit Windows operating system)

File (or Folder)

Description

Design (folder)	Includes GrapeCity.ActiveReports.Viewer.Silverlight.v7.VisualStudio.Design.4.o.dll assembly file.
License (folder)	Includes License Service assembly file.
redist (folder)	Includes native functions assembly for 64-bit machines.
ActiveReports.ReportService.asmx	Web service required for Web Site or Web Applications.
ApplicationLicenseGenerator.exe	Application License Generator setup file.
ApplicationLicenseGenerator.exe.config	License Manager setup XML configuration file.
GrapeCity.ActiveReports.Designer.exe	Standalone Designer setup file
GrapeCity.ActiveReports.Designer.exe.config	Standalone Designer setup XML configuration file
GrapeCity.ActiveReports.Imports.exe	ActiveReports Developer Import application setup file.
GrapeCity.ActiveReports.Imports.exe.config	ActiveReports Developer Import application setup XML configuration file.
GrapeCity.ActiveReports.Imports.Win.exe	ActiveReports Developer Import wizard setup file.
GrapeCity.ActiveReports.Imports.Win.exe.config	ActiveReports Developer Import wizard setup XML configuration file.
GrapeCity.ActiveReports.ThemeEditor.exe	ActiveReports Developer Theme Editor setup file.
GrapeCity.ActiveReports.ThemeEditor.exe.config	ActiveReports Developer Theme Editor setup XML configuration file.
GrapeCity.ActiveReports.Viewer.exe	Standalone ActiveReports Developer Viewer setup file.
GrapeCity.ActiveReports.Viewer.exe.config	Standalone ActiveReports Developer Viewer setup XML configuration file.
ReportDesigner.Switcher.exe	Report Designer Switcher setup file.
ReportDesigner.Switcher.exe.config	Web Key Generator setup XML configuration file.
WebKeyGenerator.exe	Web Key Generator setup file.
WebKeyGenerator.exe.config	Web Key Generator setup XML configuration file.
DocumentFormat.OpenXml.dll	OpenXML assembly file.
GrapeCity.ActiveReports.Calendar.v7.dll	Calendar control assembly file.
GrapeCity.ActiveReports.Chart.v7.dll	Chart control assembly file.
GrapeCity.ActiveReports.Dashboard.v7.dll	ActiveReports Dashboard assembly file.
GrapeCity.ActiveReports.Design.Win.v7.dll	Windows Designer assembly file.
GrapeCity.ActiveReports.Diagnostics.v7.dll	ActiveReports Diagnostics assembly file.
GrapeCity.ActiveReports.Document.v7.dll	Document assembly file.
GrapeCity.ActiveReports.Export.Document.v7.dll	Document Export assembly file.
GrapeCity.ActiveReports.Export.Html.v7.dll	HTML Export assembly file.
GrapeCity.ActiveReports.Export.Excel.v7.dll	Excel Export assembly file.
GrapeCity.ActiveReports.Export.Image.Unsafe.v7.dll	Image Export assembly file. (Unsafe version)
GrapeCity.ActiveReports.Export.Image.v7.dll	Image Export assembly file.
GrapeCity.ActiveReports.Export.Pdf.v7.dll	PDF Export assembly file.
GrapeCity.ActiveReports.Export.Rdf.v7.dll	RDF Export assembly file.
GrapeCity.ActiveReports.Export.Word.v7.dll	Word Export assembly file.
GrapeCity.ActiveReports.Export.Xaml.v7.dll	XAML Export assembly file.
GrapeCity.ActiveReports.Export.Xml.v7.dll	XML Export assembly file.
GrapeCity.ActiveReports.Extensibility.v7.dll	ActiveReports Extensibility assembly file.

GrapeCity.ActiveReports.Imports.Access.v7.dll	Microsoft Access Import assembly file.
GrapeCity.ActiveReports.Imports.Crystal.v7.dll	Crystal Reports Import assembly file.
GrapeCity.ActiveReports.Interop.v7.dll	Native functions assembly file.
GrapeCity.ActiveReports.OracleClient.v7.dll	Oracle Client assembly file.
GrapeCity.ActiveReports.Serializer.v7.dll	Serializer assembly file.
GrapeCity.ActiveReports.v7.dll	Run time engine assembly file.
GrapeCity.ActiveReports.Viewer.Silverlight.v7.dll	Silverlight Viewer assembly file.
GrapeCity.ActiveReports.Viewer.Win.v7.dll	Windows Viewer assembly file.
GrapeCity.ActiveReports.VisualStudio.v7.dll	Visual Studio assembly file.
GrapeCity.ActiveReports.Web.Design.v7.dll	Web Designer assembly file.
GrapeCity.ActiveReports.Web.v7.dll	Web assembly file.

C:\Program Files\Common Files\ComponentOne\Components (C:\Program Files (x86)\Common Files\ComponentOne\Components on a 64-bit Windows operating system)

File (or Folder)

GrapeCity.LicenseManager.exe
GrapeCity.LicenseManager.exe.config

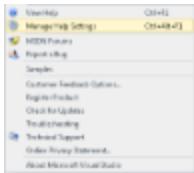
Description

License Manager setup file.
License Manager setup XML configuration file.

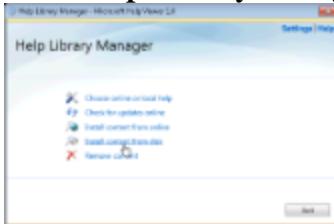
Installing Help in Visual Studio 2010

You can integrate ActiveReports Help into Visual Studio 2010 on your computer and have access to the ActiveReports Developer Guide any time you work in Visual Studio 2010.

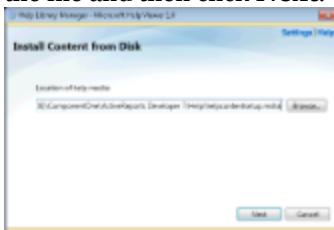
1. From the Visual Studio **Help** menu, select **Manage Help Settings**.



2. In the **Help Library Manager** window that appears, click **Install content from disk**.



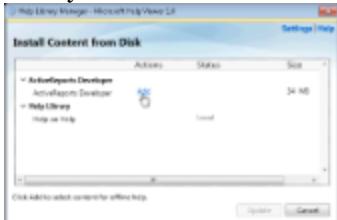
3. Click the **Browse** button to navigate to the **ActiveReports Developer 7 Help** manifest file, click **Open** to add the file and then click **Next**.



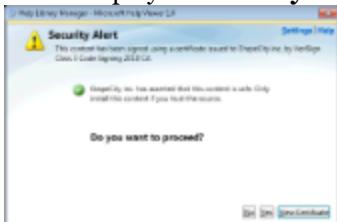
Note: By default, the file (helpcontentsetup.msha) is located at C:\Program

Files\ComponentOne\ActiveReports Developer 7\Help Or C:\Program Files (x86)\ComponentOne\ActiveReports Developer 7\Help (on a 64-bit Windows operating system).

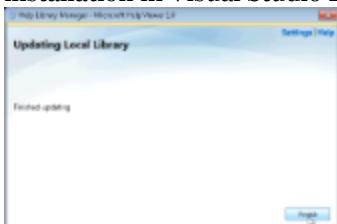
4. Click the **Add** action next to ActiveReports Developer and then click the **Update** button to start updating local library.



5. In the displayed **Security Alert** dialog, click **Yes**.



6. On the **Updating Local Library** page that appears, click **Finish** to complete the ActiveReports Developer Help installation in Visual Studio 2010.



 **Note:** If you still receive the 404 error message when you try to open the ActiveReports Developer Help, we recommend that you restart the Help Library Agent.

Side-by-Side Installation

Once ActiveReports 7 is installed on your system, it becomes the primary report designer. This means that when Visual Studio opens one of our proprietary file types, RPX or RDLX, it uses the ActiveReports 7 version of the following Visual Studio integrated features:

- Integrated report designer
- Report menu
- ActiveReports toolbar
- Report Explorer (or Data Explorer in Data Dynamics Reports)
- Toolbox tabs

The included ReportDesigner.Switcher tool allows you to change which packages are registered in Visual Studio. You can change between ActiveReports 6 and ActiveReports 7 for the RPX designer, and between ActiveReports 7 and Data Dynamics Reports for the RDLX designer.



Tip: You can still access some of the integrated features even if they are hidden.

- Alternate Report menus are visible, but disabled unless you are on the right type of report.
- Right-click in the Visual Studio toolbar area to select which toolbars to show.
- The Visual Studio **View** menu, under **Other Windows**, lets you select Report Explorer versions.

- Toolbox tabs are still visible for other versions, but ones that do not work with the current designer are disabled.

With ActiveReports 7 as the primary designer, if you open an existing code-based report from ActiveReports 6, Visual Studio knows to use the ActiveReports 6 designer, but if you open an xml-based report from ActiveReports 6 (RPX) or Data Dynamics Reports (RDLX), Visual Studio sees the proprietary file extension and opens it with the most recently registered package, in this case, ActiveReports 7.

In order to work with ActiveReports 6 RPX reports, you must run the Switcher tool and change the Primary RPX designer to ActiveReports 6. If you do not, the reports open in the ActiveReports 7 designer, and the toolbar does not work, you might inadvertently use features from ActiveReports 7 that do not exist in ActiveReports 6, thus breaking the reports when they try to run with the ActiveReports 6 DLLs.

Switching between ActiveReports 7 and ActiveReports 6

- Close all instances of Visual Studio.
- In ...\\Common Files\\ComponentOne\\ActiveReports Developer 7, double-click **ReportDesigner.Switcher.exe** to run the switcher tool.
- In the dialog that appears, under Primary RPX designer, choose the radio button for the product you want to use and click **OK**.

Similarly, in order to work with Data Dynamics Reports RDLX reports, you must run the Switcher tool and change the Primary RDLX designer to Data Dynamics Reports.

Switching between ActiveReports 7 and Data Dynamics Reports

- Close all instances of Visual Studio.
- In ...\\Common Files\\ComponentOne\\ActiveReports Developer 7, double-click **ReportDesigner.Switcher.exe** to run the switcher tool.
- In the dialog that appears, under Primary RDLX designer, choose the radio button for the product you want to use and click **OK**.

ComponentOne Copyright Notice

Information in this document, including URL and other Internet 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 ComponentOne, a division of GrapeCity.

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 ComponentOne is given credit within the distributed documentation.

ActiveReports and the ActiveReports logo are registered trademarks of ComponentOne, a division of GrapeCity.

All other trademarks are the property of their respective owners.

End User License Agreement

The End-User license agreement is available online at <http://www.componentone.com/SuperPages/DevToolsEULA/>.

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 ComponentOne, a division of GrapeCity, if you have any

questions about this license.

.NET Framework Client and Full Profile Versions

All ActiveReports Developer assemblies are compliant with .NET Framework 3.5 Full profile and .NET Framework 4.0 Full profile.

The following assemblies are compliant with .NET Framework 3.5 Client Profile and .NET Framework 4.0 Client Profile:

File	Description
GrapeCity.ActiveReports.v7.dll	Run-time engine assembly file.
GrapeCity.ActiveReports.Chart.v7.dll	Chart control assembly file.
GrapeCity.ActiveReports.Document.v7.dll	Document assembly file.
GrapeCity.ActiveReports.Interop.v7.dll	Native functions assembly file.
GrapeCity.ActiveReports.Export.Pdf.v7.dll	PDF Export assembly file.
GrapeCity.ActiveReports.Export.Word.v7.dll	RTF Export assembly file.
GrapeCity.ActiveReports.Export.Xml.v7.dll	Text Export assembly file.
GrapeCity.ActiveReports.Export.Image.v7.dll	TIFF Export assembly file.
GrapeCity.ActiveReports.Viewer.Win.v7.dll	Viewer assembly file.
GrapeCity.ActiveReports.Export.Excel.v7.dll	Microsoft® Excel® Export assembly file.
GrapeCity.ActiveReports.Extensibility.v7.dll	Extensibility assembly file.

The following assemblies are not compliant with .NET Framework 3.5 Client Profile and .NET Framework 4.0 Client Profile:

File	Description
GrapeCity.ActiveReports.Design.Win.v7.dll	Designer assembly file.
GrapeCity.ActiveReports.Export.Html.v7.dll	HTML Export assembly file.
GrapeCity.ActiveReports.Web.v7.dll	Web assembly file.
GrapeCity.ActiveReports.OracleClient.v7.dll	Oracle Client assembly file.

The **End User Report Designer**, the **WebViewer** control and the **HTML Export** filter require the full profile.

Redistributable Files

ActiveReports Developer 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
- GrapeCity.ActiveReports.Calendar.v7.dll
- GrapeCity.ActiveReports.Chart.v7.dll
- GrapeCity.ActiveReports.Dashboard.v7.dll
- GrapeCity.ActiveReports.Design.Win.v7.dll
- GrapeCity.ActiveReports.Diagnostics.v7.dll

- GrapeCity.ActiveReports.Document.v7.dll
- GrapeCity.ActiveReports.Export.Document.v7.dll
- GrapeCity.ActiveReports.Export.Html.v7.dll
- GrapeCity.ActiveReports.Export.Excel.v7.dll
- GrapeCity.ActiveReports.Export.Image.Unsafe.v7.dll
- GrapeCity.ActiveReports.Export.Image.v7.dll
- GrapeCity.ActiveReports.Export.Pdf.v7.dll
- GrapeCity.ActiveReports.Export.Rdf.v7.dll
- GrapeCity.ActiveReports.Export.Word.v7.dll
- GrapeCity.ActiveReports.Export.Xaml.v7.dll
- GrapeCity.ActiveReports.Export.Xml.v7.dll
- GrapeCity.ActiveReports.Extensibility.v7.dll
- GrapeCity.ActiveReports.Imports.Access.v7.dll
- GrapeCity.ActiveReports.Imports.Crystal.v7.dll
- GrapeCity.ActiveReports.Interop.v7.dll
- GrapeCity.ActiveReports.OracleClient.v7.dll
- GrapeCity.ActiveReports.Serializer.v7.dll
- GrapeCity.ActiveReports.v7.dll
- GrapeCity.ActiveReports.Viewer.Silverlight.v7.dll
- GrapeCity.ActiveReports.Viewer.Win.v7.dll
- GrapeCity.ActiveReports.VisualStudio.v7.dll
- GrapeCity.ActiveReports.Web.Design.v7.dll
- GrapeCity.ActiveReports.Web.v7.dll
- GrapeCity.ActiveReports.Interop64.v7.dll
- GrapeCity.ActiveReports.Imports.exe
- GrapeCity.ActiveReports.Imports.Win.exe
- Grapecity.ActiveReports.Flash.v7.swf
- Grapecity.ActiveReports.Flash.v7.Resources.swf
- Themes\FluorescentBlue.swf
- Themes\Office.swf
- Themes\OliveGreen.swf
- Themes\Orange.swf
- Themes\VistaAero.swf
- Themes\WindowsClassic.swf
- Themes\XP.swf

 **Note:** See **Installed Files** for the location of the files listed above.

License Your ActiveReports

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

License Types

ActiveReports Developer provides three licensing options to choose from.

License Type	Description
--------------	-------------

Evaluation	No product key is required. All evaluation banners are displayed. There is an initial 30 day time-limit for product use but it can be extended after contacting the sales department.
Standard	Product key for the standard edition is required. It offers basic report functionality without any evaluation banners.
Professional	Product key for professional edition is required. All reporting functionality and controls can be accessed with this license without any evaluation banners.

See **ActiveReports Editions** to understand the difference between Standard and Professional Editions.

License Key Types

 **Note:** If you have purchased ActiveReports Developer, your product key should have been emailed to you at the time of purchase. If you do not have a product key, you can retrieve it from sales@componentone.com.

ActiveReports Developer offers different key types with varying functionality in the way the product works.

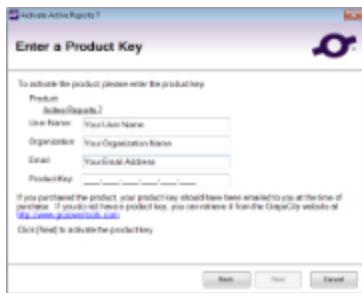
License Key Type	Description
Trial	This key type shows all evaluation banners and the product stops functioning after 30 days from the date of installation. Users can request a new key from the Sales department to grant them an additional 30 days.
Standard	This key type is given to users who purchase ActiveReports Developer Standard Edition or those who are upgrading from a previous version of ActiveReports Standard Edition.
Professional	This key type is given to users who purchase ActiveReports Developer Professional Edition or those who are upgrading from a previous version of ActiveReports Professional Edition.

To license ActiveReports Developer on installation or license a trial without reinstalling

1. From the Start menu, go to the **All Programs > ComponentOne > License Manager**.
2. In the GrapeCity License Manager window that appears, under **Action** click **Activate**.



3. On the **Activate [Active Reports 7]** screen that appears, click the Next button.
4. In the **Enter a Product Key** screen that appears next, enter the following information:
 - **User Name:** Enter your name here.
 - **Organization:** Enter your company name here.
 - **Email:** Enter your e-mail address here.
 - **Product Key:** Enter the product key exactly as you received it from ComponentOne, including any capital letters. When you enter the product key, a green check mark appears next to this field to indicate a valid key.



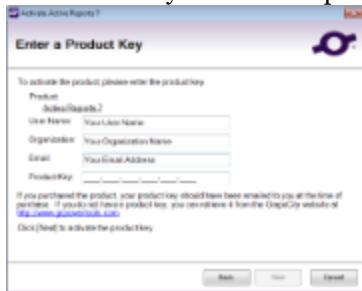
- Click the **Next** and then the **Finish** button to complete the licensing process.

To license ActiveReports Developer on a machine without Internet Connection

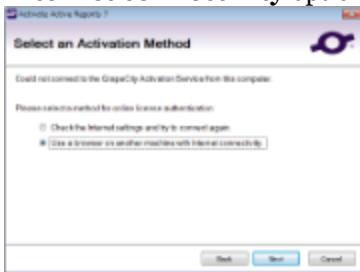
- At the time of installation, on the last screen a check box that states **Run license manager** appears. Select this checkbox and click the **Close** button to complete the installation.
- In the GrapeCity License Manager window that appears, under the **Action** field, click **Activate**.



- On the **Activate [Active Reports 7]** screen that appears, click the **Next** button.
- In the **Enter a Product Key** screen that appears next, enter the following information:
 - User Name:** Enter your name here.
 - Organization:** Enter your company name here.
 - Email:** Enter your e-mail address here.
 - Product Key:** Enter the product key exactly as you received it from ComponentOne, including any capital letters. When you enter the product key, a green check mark appears next to this field to indicate a valid key.



- Click the **Next** button to authenticate the license.
- If your machine does not have an internet connection, select the **Use a browser on another machine with Internet connectivity** option from the following screen and click the **Next** button.

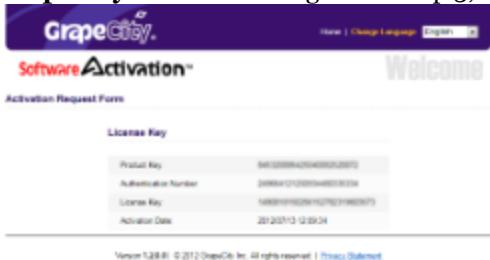


- From the **Activate using the GrapeCity web site** screen that appears, copy the Product Key and

Authentication Number.



8. On another machine with an internet connection, go to <https://sas.grapecity.com/activation>. Remember to not close the activation dialog on your original machine till the activation process is complete.
9. Enter the Product Key and Authentication Number you copied in step 7 on this website.
10. Click the **Send Request** button to generate a license key.
11. Copy the license key from the web page that looks like the following image and in the **Activate using GrapeCity web site** dialog under step 3, enter the key.



12. Click the **Next** and then the **Finish** button to complete the licensing process.

To license Windows Forms projects made on the trial version

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

1. Open the project in Microsoft Visual Studio.
2. Go to the Visual Studio **Build** menu and select **Rebuild Solution**. The executable application is now licensed, and no nag screens or evaluation banners appear when you run it. You can distribute the application to unlicensed machines and no nag screens or evaluation banners appear.

To license Web Forms projects made on the trial version

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

1. Open the project in Microsoft Visual Studio.
 2. Open the Visual Studio **Build** menu and select **Rebuild Solution**.
- Note:** For licensing Web Site applications, open the Visual Studio **Build** menu and select **Build Runtime Licenses** to create the App_Licenses.dll file.
3. The web application is now licensed, no evaluation banners appear when you run it. You can distribute the Web application to unlicensed machines and no evaluation banners appear.

To upgrade or downgrade a license

If you want to change your ActiveReports Developer license type you need to do one of the following:

Upgrade from a Standard to a Professional License:

1. From the Start menu, go to the **All Programs > ComponentOne > License Manager**.
2. In the GrapeCity License Manager window that appears, under Upgrade/Downgrade click **Upgrade to Professional License**.
3. Follow the activation steps from step 3 of **To license an ActiveReports Developer Trial without reinstalling** to upgrade.

Downgrade from a Professional to a Standard License:

1. From the Start menu, go to the **All Programs > ComponentOne > License Manager**.
2. In the GrapeCity License Manager window that appears, under Upgrade/Downgrade click **Downgrade to Standard License**.
3. In the Deactivate the Product Key screen that appears select the **Next** button.
4. Confirm the Product screen appears. Confirm that the correct product is getting downgraded and click the **Next** button.
5. Deactivation Successful screen appears with the **Product Name** as Active Reports 7 and the **Current Status** as Standard License.

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

The licenses.licx file must contain the following references to the ActiveReports Developer version and the reference to the Viewer control:

Standard Edition:

Paste INSIDE the licenses.licx file. Replace Version=x.x.xxxx.x with the actual ActiveReports Developer version.

(Reference to GrapeCity.ActiveReports.SectionReport is added only for code-based section report templates)

GrapeCity.ActiveReports.SectionReport, GrapeCity.ActiveReports.v7, Version=x.x.xxxx.x, Culture=neutral, PublicKeyToken=cc4967777c49a3ff

GrapeCity.ActiveReports.Viewer.Win.Viewer, GrapeCity.ActiveReports.Viewer.Win.v7, Version=x.x.xxxx.x, Culture=neutral, PublicKeyToken=cc4967777c49a3ff

Professional Edition:

Paste INSIDE the licenses.licx file. Replace Version=x.x.xxxx.x with the actual ActiveReports Developer version.

(Reference to GrapeCity.ActiveReports.SectionReport is added only for code-based section report templates)

GrapeCity.ActiveReports.SectionReport, GrapeCity.ActiveReports.v7, Version=x.x.xxxx.x, Culture=neutral, PublicKeyToken=cc4967777c49a3ff

GrapeCity.ActiveReports.Viewer.Win.Viewer, GrapeCity.ActiveReports.Viewer.Win.v7, Version=x.x.xxxx.x, Culture=neutral, PublicKeyToken=cc4967777c49a3ff

GrapeCity.ActiveReports.Web.WebViewer, GrapeCity.ActiveReports.Web.v7, Version=x.x.xxxx.x, Culture=neutral, PublicKeyToken=cc4967777c49a3ff

GrapeCity.ActiveReports.Design.Designer, GrapeCity.ActiveReports.Design.Win.v7, Version=x.x.xxxx.x, Culture=neutral, PublicKeyToken=cc4967777c49a3ff

GrapeCity.ActiveReports.Export.Pdf.Section.PdfExport, GrapeCity.ActiveReports.Export.Pdf.v7, Version=x.x.xxxx.x, Culture=neutral, PublicKeyToken=cc4967777c49a3ff

 **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.

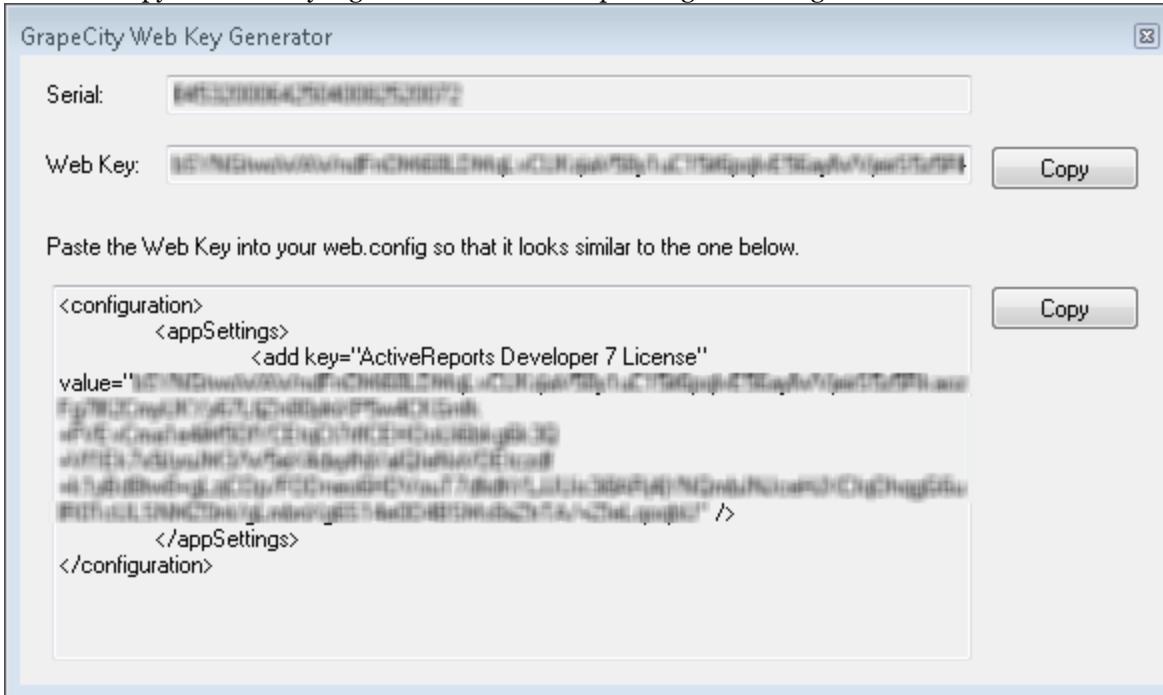
To create the Web Key with the Web Key Generator utility

For your medium trust and Windows Azure projects with ActiveReports Developer, you may need to generate a Web Key by using the Web Key Generator utility on a machine with licensed ActiveReports Developer.

- From the Start Menu, go to **All Programs > ComponentOne > ActiveReports Developer > Generate WebKey** and run the **Web Key Generator** utility.

 **Note:** You can find the WebKeyGenerator.exe in the\Common files\ComponentOne\ActiveReports Developer 7 folder.

- In the dialog that appears, copy the Web Key by clicking **Copy**. We recommend that you use the second **Copy** button to copy the Web Key together with the corresponding web.config section.



- Paste the Web Key into the web.config file of your project between the opening and closing <configuration> tags to remove the licensing message. The web.config key looks like the following.

XML code. Paste INSIDE the Web.config file

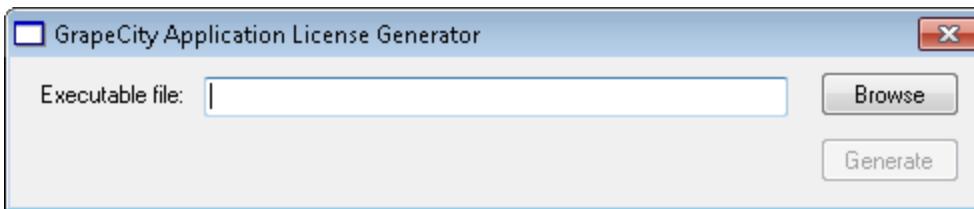
```
<configuration>
  <appSettings>
    <add key=" ActiveReports Developer 7 License" value="Generated Web Key" />
  </appSettings>
</configuration>
```

 **Note:** If you see the message "Your computer is not currently licensed" in the **Web Key Generator** dialog, please license your machine.

To license a class library project

You can license your ActiveReports Developer class library project using the Application License Generator utility.

- Ensure that ActiveReports Developer is licensed on the machine by following the steps above in the "**To license ActiveReports Developer on a machine during installation**" or "**To license an ActiveReports Developer trial without reinstallation**" sections.
- Run the **ApplicationLicenseGenerator.exe** from the ...\\Common Files\\ComponentOne\\ActiveReports Developer 7 folder.



3. Click the **Browse** button and select the compiled dll that requires licensing.
4. Click the **Generate** button.

Distribute the generated file <AssemblyName>.GrapeCity.Licenses.dll along with the application.

To remove an Invalid license message

If your license key is invalid or corrupt, you may get an invalid license message.

To remove the message, do any one of the following:

- Run the setup to re-install and authenticate the license.
- Open the GrapeCity License Manager utility by clicking **Run the License Manger (activate/deactivate)** link in this error message and click Activate.

Follow the licensing steps from **To license an ActiveReports Developer Trial without reinstalling** to license your product and remove the error message.

To deactivate an ActiveReports Developer license

You can deactivate your ActiveReports Developer license and go back to a Trial License.

1. From the Start menu, go to the **All Programs > ComponentOne > License Manager**.
2. In the GrapeCity License Manager window that appears, under Action click **Deactivate**.
3. In the Deactivate [Active Reports 7] screen that appears select the **Next** button.
4. Confirm the Product screen appears. Confirm that the correct product is getting downgraded and click the **Next** button.
5. Deactivation Successful screen appears with the **Product Name** as Active Reports 7 and the **Current Status** as Trial License (number of days left).

Upgrading Reports

ActiveReports Developer 7 allows you to upgrade your reports from other versions of ActiveReports and Data Dynamics Reports.

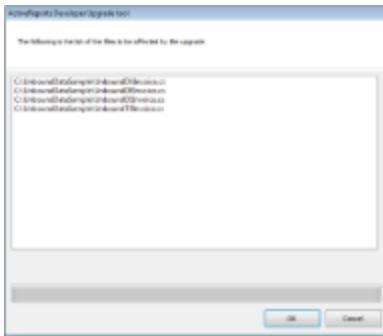
Upgrading ActiveReports 6 and Data Dynamics Reports Projects

You can automatically update reports and corresponding reference assemblies using the ActiveReports Developer Upgrade tool, and then handle any additional errors in the code.

Important: Be sure to create a backup for your project before starting the upgrade process.

To upgrade project references and reports

1. In Visual Studio, open an existing ActiveReports 6 or Data Dynamics Reports project that you want to upgrade.
2. From the Visual Studio **Tools** menu, select **Convert to ActiveReports 7**.
3. In the ActiveReports Developer Upgrade tool window that appears, you can see a list of report files to be converted.



4. Click **OK** to upgrade the project.

Once the tool upgrades the report files, notice that all of the ActiveReports 6 or Data Dynamics Reports assemblies in the Solution Explorer are replaced with ActiveReports Developer assembly references.

To handle errors in code

The ActiveReports Developer Upgrade tool upgrades the reports and corresponding reference assemblies, but the upgraded assembly names break the code.

Note: The error messages in a just-upgraded project may seem a bit odd if the project retains some of the old references in memory. To work around this Visual Studio issue, you can close and reopen the project for more understandable error messages.

1. With the upgraded project open in Visual Studio, from the **View** menu, select **Error List** to get the list of errors to be fixed in code.
2. In the Error List window, double-click each error in turn to jump to the code where you can fix the error.

Examples of code that might break and how to fix it:

- Update the Viewer control by replacing `DataDynamics.ActiveReports.Viewer.Viewer` with `GrapeCity.ActiveReports.Viewer.Win.Viewer`.
- Replace existing property names with new property names. For example, `MultiplePageCols` property is changed to `MultiPageCols`.

Breaking Changes

When you upgrade reports from previous versions of ActiveReports or Data Dynamics Reports, there are several breaking changes.

Control Changes

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 in a path like the following: `C:\Program Files (x86)\ComponentOne\ActiveReports Developer 7`.

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 7 Web Service**, or manually by copying it from `C:\Program Files (x86)\Common Files\ComponentOne\ActiveReports Developer 7`.

The **Viewer** control no longer has Annotations turned on by default. To enable Annotations, set the **AnnotationToolbarVisible** property of the Viewer control to **True**.

The **Toolbar** is now a Windows ToolStrip. Please see the [MSDN ToolStrip Class](#) for more information.

Classes in Different Namespaces

Some classes have moved to different namespaces from previous versions of ActiveReports. Drop down the table below to see some of the most commonly used classes that are in new namespaces.

Classes that are in new namespaces

Class Name	New Namespace	Former Namespace
Report	GrapeCity.ActiveReports.PageReportModel	DataDynamics.Reports.ReportObjectModel
SectionReport (formerly ActiveReport)	GrapeCity.ActiveReports	DataDynamics.ActiveReports
PageReport (formerly ReportDefinition)	GrapeCity.ActiveReports	DataDynamics.Reports
SectionDocument (formerly Document)	GrapeCity.ActiveReports.Document	DataDynamics.ActiveReports.Document
SystemPrinter	GrapeCity.ActiveReports	DataDynamics.ActiveReports.Interop
Printer	GrapeCity.ActiveReports.Extensibility.Printing	DataDynamics.ActiveReports.Document
Exports		
HtmlExport	GrapeCity.ActiveReports.Export.Html.Section	DataDynamics.ActiveReports.Export.Html
PdfExport	GrapeCity.ActiveReports.Export.Pdf.Section	DataDynamics.ActiveReports.Export.Pdf
PdfSignature	GrapeCity.ActiveReports.Export.Pdf.Section.Signing	DataDynamics.ActiveReports.Export.Pdf.Signing
PdfStamp	GrapeCity.ActiveReports.Export.Pdf.Section.Signing	DataDynamics.ActiveReports.Export.Pdf.Signing
RtfExport	GrapeCity.ActiveReports.Export.Word.Section	DataDynamics.ActiveReports.Export.Rtf
TextExport	GrapeCity.ActiveReports.Export.Xml.Section	DataDynamics.ActiveReports.Export.Text
TiffExport	GrapeCity.ActiveReports.Export.Image.Tiff.Section	DataDynamics.ActiveReports.Export.Tiff
XlsExport	GrapeCity.ActiveReports.Export.Excel.Section	DataDynamics.ActiveReports.Export.Xls
ImageRenderingExtension	GrapeCity.ActiveReports.Export.Image.Page	DataDynamics.Reports.Rendering.Graphics
HtmlRenderingExtension	GrapeCity.ActiveReports.Export.Html.Page	DataDynamics.Reports.Rendering.Html
PdfRenderingExtension	GrapeCity.ActiveReports.Export.Pdf.Page	DataDynamics.Reports.Rendering.Pdf
XmlRenderingExtension	GrapeCity.ActiveReports.Export.Xml.Page	DataDynamics.Reports.Rendering.Xml
WordRenderingExtension	GrapeCity.ActiveReports.Export.Word.Page	DataDynamics.Reports.Rendering.Word
ExcelTransformationDevice	GrapeCity.ActiveReports.Export.Excel.Page	DataDynamics.Reports.Rendering.Excel
Report Controls		
All section report controls	GrapeCity.ActiveReports.SectionReportModel	DataDynamics.ActiveReports
All page report controls	GrapeCity.ActiveReports.PageReportModel	DataDynamics.Reports.ReportObjectModel

Namespace Changes and Restructuring

Some of the changes that are not caught by the upgrade tool may cause some issues with 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.

ActiveReports6 is now GrapeCity.ActiveReports.v7

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

ActiveReports 6 Namespace

DataDynamics.ActiveReports

DataDynamics.ActiveReports.DataSources

DataDynamics.ActiveReports.Interop

DataDynamics.ActiveReports.Options

ActiveReports.Chart is now GrapeCity.ActiveReports.Chart.v7

ActiveReports Developer 7 Namespace

- GrapeCity.ActiveReports
- GrapeCity.ActiveReports.SectionReportModel
- GrapeCity.ActiveReports.Data

GrapeCity.ActiveReports.Data

GrapeCity.ActiveReports

GrapeCity.ActiveReports.SectionReportModel

ActiveReports 6 Namespace

DataDynamics.ActiveReports.Chart
DataDynamics.ActiveReports.Chart.Annotations
DataDynamics.ActiveReports.Chart.Graphics

ActiveReports Developer 7 Namespace

GrapeCity.ActiveReports.Chart
GrapeCity.ActiveReports.Chart.Annotations
GrapeCity.ActiveReports.Chart.Graphics

ActiveReports.Design6 is now GrapeCity.ActiveReports.Design.Win.v7

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

ActiveReports 6 Namespace

DataDynamics.ActiveReports.Design
DataDynamics.ActiveReports.Design.ReportExplorer
DataDynamics.ActiveReports.Design.Toolbox

ActiveReports Developer 7 Namespace

GrapeCity.ActiveReports.Design
GrapeCity.ActiveReports.ReportExplorer
GrapeCity.ActiveReports.Design.Toolbox

ActiveReports.Document is now GrapeCity.ActiveReports.Document.v7

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

ActiveReports 6 Namespace

DataDynamics.ActiveReports
DataDynamics.ActiveReports.Document

DataDynamics.ActiveReports.Export
DataDynamics.ActiveReports.Export.Html
DataDynamics.ActiveReports.Document.Annotations

ActiveReports Developer 7 Namespace

GrapeCity.ActiveReports

- GrapeCity.ActiveReports.Document
- GrapeCity.ActiveReports.Document.Section
- GrapeCity.ActiveReports.Extensibility.Printing(GrapeCity.ActiveReports.Extensibility.v7)

GrapeCity.ActiveReports.Export
GrapeCity.ActiveReports.Export.Html
GrapeCity.ActiveReports.Document.Section.Annotations

ActiveReports.HtmlExport is now GrapeCity.ActiveReports.Export.Html.v7

ActiveReports 6 Namespace

DataDynamics.ActiveReports.Export.Html

ActiveReports Developer 7 Namespace

GrapeCity.ActiveReports.Export.Html.Section

ActiveReports.PdfExport is now GrapeCity.ActiveReports.Export.Pdf.v7

ActiveReports 6 Namespace

DataDynamics.ActiveReports.Export.Pdf
DataDynamics.ActiveReports.Export.Pdf.Signing

ActiveReports Developer 7 Namespace

GrapeCity.ActiveReports.Export.Pdf.Section
GrapeCity.ActiveReports.Export.Pdf.Section.Signing

ActiveReports.RtfExport is now GrapeCity.ActiveReports.Export.Word.v7

ActiveReports 6 Namespace

DataDynamics.ActiveReports.Export.Rtf

ActiveReports Developer 7 Namespace

GrapeCity.ActiveReports.Export.Word.Section

ActiveReports.Silverlight is now GrapeCity.ActiveReports.Viewer.Silverlight.v7

ActiveReports 6 Namespace

DataDynamics.ActiveReports

ActiveReports Developer 7 Namespace

GrapeCity.ActiveReports

ActiveReports.TextExport is now GrapeCity.ActiveReports.Export.Xml.v7

ActiveReports 6 Namespace

DataDynamics.ActiveReports.Export.Text

ActiveReports Developer 7 Namespace

GrapeCity.ActiveReports.Export.Xml.Section

ActiveReports.TiffExport is now GrapeCity.ActiveReports.Export.Image.v7

ActiveReports 6 Namespace

DataDynamics.ActiveReportsExport.Tiff

ActiveReports Developer 7 Namespace

GrapeCity.ActiveReports.Export.Image.Tiff.Section

ActiveReports.Viewer6 is now GrapeCity.ActiveReports.Viewer.Win.v7

- 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 **TargetView** enumeration now has two enumeration values (Primary and Secondary).

ActiveReports 6 Namespace ActiveReports Developer 7 Namespace

DataDynamics.ActiveReports.Toolbar	The viewer now uses Visual Studio ToolStrips. Please see MSDN ToolStrip Class for more information.
DataDynamics.ActiveReports.Viewer	<ul style="list-style-type: none">GrapeCity.ActiveReports.Viewer.WinGrapeCity.Viewer.Common

 **Note:** GrapeCity.ActiveReports.Viewer.Win.v7.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.v7

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

ActiveReports 6 Namespace

DataDynamics.ActiveReports.Web
DataDynamics.ActiveReports.Web.Controls
DataDynamics.ActiveReports.Web.ExportOptions
DataDynamics.ActiveReports.Web.Handlers

ActiveReports Developer 7 Namespace

GrapeCity.ActiveReports.Web
GrapeCity.ActiveReports.Web.Controls
GrapeCity.ActiveReports.Web.ExportOptions
GrapeCity.ActiveReports.Web.Handlers

ActiveReports.XlsExport is now GrapeCity.ActiveReports.Export.Excel.v7

ActiveReports 6 Namespace

DataDynamics.ActiveReports.Export.Xls
DataDynamics.SpreadBuilder
DataDynamics.SpreadBuilder.Cells
DataDynamics.SpreadBuilder.Imaging
DataDynamics.SpreadBuilder.Printing
DataDynamics.SpreadBuilder.Style

ActiveReports Developer 7 Namespace

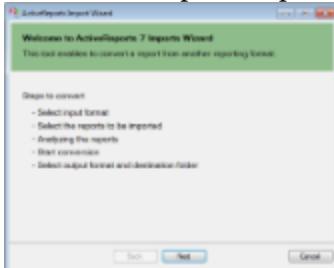
GrapeCity.ActiveReports.Export.Excel.Section
GrapeCity.SpreadBuilder
GrapeCity.SpreadBuilder.Cells
GrapeCity.SpreadBuilder.Imaging
GrapeCity.SpreadBuilder.Printing
GrapeCity.SpreadBuilder.Style

Converting Crystal Reports/MS Access Reports

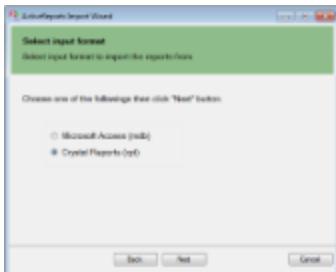
You can convert a Crystal Reports report or a Microsoft Access report to an ActiveReports format by running the ActiveReports Developer Import Wizard.

Running the ActiveReports Developer Import Wizard

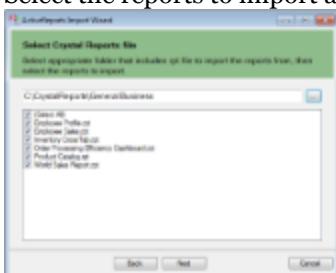
- From the Start Menu, go to **All Programs > ComponentOne > ActiveReports Developer > ActiveReports Developer Import**.
- In the ActiveReports Import Wizard that appears, click **Next** to proceed to the conversion process.



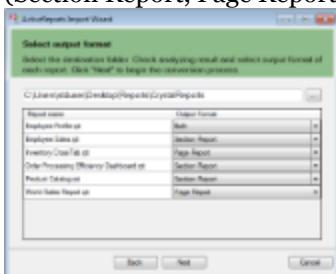
- Choose **Microsoft Access (mdb)** or **Crystal Reports (rpt)** as the input format and click **Next** to convert the selected report.



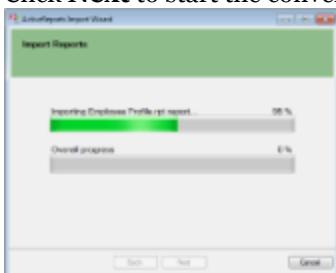
4. Browse to the location that contains the Microsoft Access file or Crystal Reports using the ellipsis button. Once you select the location, a list of available reports appear in the dialog.
5. Select the reports to import and click **Next** to analyze the selected reports.



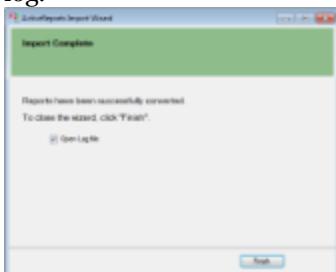
6. Use the ellipsis button to select a destination folder to store the converted reports. Also select an output format (Section Report, Page Report or Both) for each report in the Output Format column.



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, whether a report is imported as a Fixed Page Layout (FPL) or Continuous Page Layout (CPL), depends on the following factors:

- If a report has a single detail section it is imported as a Fixed Page Layout.
- If a report has a SubReport control it is imported as a Continuous Page Layout.
- If a report has a Matrix control and its layout is composed of multiple sections it is imported as a Continuous Page Layout.

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

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

Converting Crystal Reports

To convert Crystal Reports into ActiveReports format, 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	Professional, Premium, Ultimate System	Crystal Reports for Visual Studio 2010	13.0.2000.0

Crystal Report controls are converted in ActiveReports as follows:

Crystal Report	Section	Page 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.

Converting Microsoft Access Reports

To convert Microsoft® Access® reports into ActiveReports format, you must have Access 97, 2000, 2002, 2003 or 2007 installed on your system.

Microsoft Access report controls are converted in ActiveReports as follows:

Microsoft	Section	Page	Note
-----------	---------	------	------

Access Report	Report	Report	
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, 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 converted.
- The shadow property of a control is not imported while converting a report.
- The OLE object is not in Crystal Reports is not imported as it is treated as PictureObject in the object structure.
- In Microsoft Access reports, VBA code appears in as commented statements in script. You have to modify the code after importing.

Getting Started

Quickly begin using ActiveReports by reviewing some of the most commonly used features.

This section contains information about

Adding ActiveReports Controls

Learn how to add ActiveReports controls to the toolbox in Visual Studio.

Adding an ActiveReport to a Project

Learn how to add an ActiveReport to a Visual Studio project. Also in this section, learn about the different types of reports and how to add code or script to each.

Adding a Data Source to a Report

Learn about the different ways that you can add data to each type of report, and where to find more information on each.

Viewing Reports

Learn how to preview a report at design time or view it in Windows Form, Web or Silverlight Viewers.

Adding an ActiveReports Application

Learn how to add an ActiveReports application to the Visual Studio project and avoid additional implementation rendering the report in the Viewer.

Adding ActiveReports Controls

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.

The installer generally adds the controls to the Visual Studio toolbox in an **ActiveReports 7** tab. However, if they are

removed for any reason, you can re-add them at any time.

To add the controls

1. Right-click the Visual Studio toolbox tab where you want to add ActiveReports controls and select **Choose Items**.
2. In the Choose Toolbox Items window that appears, on the .NET Framework Components tab, in the Filter textbox, enter **GrapeCity.ActiveReports**.
3. Select the check boxes next to any of the controls that you want to add to your toolbox:

<ul style="list-style-type: none">• Viewer• WebViewer• Designer• ReportExplorer• Toolbox	<ul style="list-style-type: none">• HtmlExport• PdfExport• RtfExport• TextExport• TiffExport• XlsExport
--	--
4. For the Silverlight Viewer control, go to the **Silverlight Components** tab and select **Viewer**.
5. Click **OK** to add the controls to the selected toolbox tab.

.NET Framework Version

The following features all require the .NET Framework full profile version.

- Designer control
- WebViewer control
- HTMLExport
- Oracle data provider

To ensure that you are using the full profile version in a VB project

1. From the Visual Studio **Project** menu, select **YourProject Properties**.
2. On the **Compile** tab, click the **Advanced Compile Options** button.
3. In the Advanced Compiler Settings dialog that appears, drop down the **Target framework** field and select a version that does *not* specify Client Profile.

To ensure that you are using the full profile version in a C# project

1. From the Visual Studio **Project** menu, select **YourProject Properties**.
2. On the **Application** tab, drop down the **Target framework** field and select a version that does *not* specify Client Profile.

 **Caution:** ActiveReports controls may not appear in the toolbox unless your project is using .NET 3.5 or later.

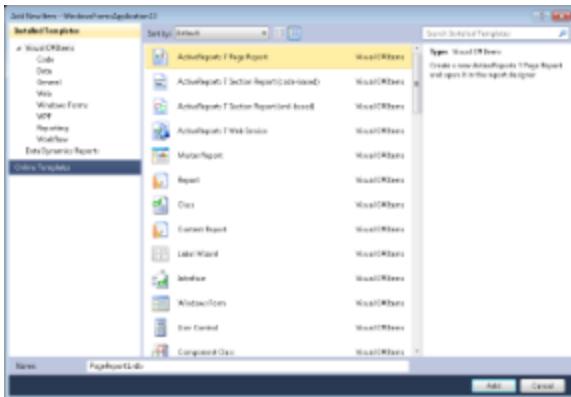
Adding an ActiveReport to a Project

To use ActiveReports in a Visual Studio project, you add one of the included report templates.

To 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**):
 - ActiveReports 7 Section Report (code-based)
 - ActiveReports 7 Section Report (xml-based)

- ActiveReports 7 Page 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.

 **Note:** When you add a report layout the Viewer assembly (GrapeCity.ActiveReports.Viewer.Win.v7.dll) is not added automatically to the project references. You may need to manually add it in your project if required.

Adding 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 Data

With page 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 to page reports, see the Walkthroughs section for step by step instructions on using **Reports with Stored Procedures**, or creating a **Custom Data Provider**.

Section Report Data

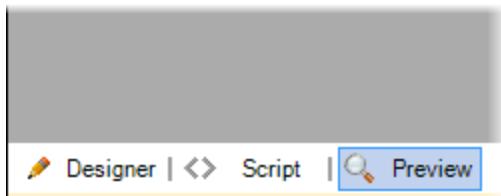
With section reports, you bind a report to any of a variety of data sources and select the data using a SQL query or XPath expression in the Data Source Dialog. You can also use code to create an unbound data source or to change the data source at run time. For more information on all of these methods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

Viewing 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, Web and Silverlight, you can view your report in any of these platforms as well in a separate viewer control. The following topics introduce all the available report viewing options.

In this section

Using the Viewer

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

ActiveReports and the Web

This section introduces the Web Viewer where you can view your report output in various types of viewers and provides key features of each viewer type.

Using the Silverlight Viewer

This section describes how to view a report in the Silverlight viewer and introduces its toolbar and features.

Using the WPF Viewer

This section describes the WPF Viewer toolbar, its additional features and how to view a report in the WPF viewer.

Using the 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.
1/13	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

	Forward	first time also enables the Forward button.
	Back to parent report	Takes you to last viewed page before you clicked the Backward button. This button is enabled once you click the Backward button.
	Default	Returns you to the parent report in a drillthrough report.
	Pan mode	Allows you to specify a default mouse pointer mode.
	Selection mode	A hand serves as the pointer that you can use to navigate the report.
	Snapshot 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.
	Toggle sidebar	Allows you to select content on the report that you can paste as an image into any application that accepts pasted images.
	Print	Displays the sidebar that includes the Thumbnails, Parameters, Document map and Search results panes.
	Galley mode	Displays the Print dialog where you can specify the printing options.
	Copy	Provides a viewer mode which removes automatic page breaks from a Continuous Page Layout (CPL) and displays data in a single page. This mode maintains page breaks you create in the report and removes only automatic page breaks.
	Find	Copies text that you select in the Selection mode to the clipboard.
	Zoom out	<div style="border: 1px solid #ccc; padding: 5px;"><p> Note: In case the GrapeCity.ActiveReports.Export.Xml.v7.dll and GrapeCity.ActiveReports.Export.Word.v7.dll are not available in GAC, you might need to add references to these assembly files to enable the viewer's Copy button.</p></div>
	Current zoom	Displays the Find dialog to find any text in the report.
	Zoom in	Decreases the magnification of your report.
	Fit width	Displays the current zoom percentage which can also be edited.
	Fit page	Increases the magnification of your report.
		Fits the width of the page according to viewer dimensions.
		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.
		<p>Caution: Refresh button gets disabled when you load a section report in the Viewer control through any of the following:</p> <ul style="list-style-type: none">• 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.

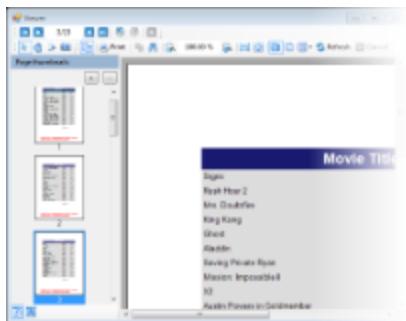
Viewer Sidebar

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

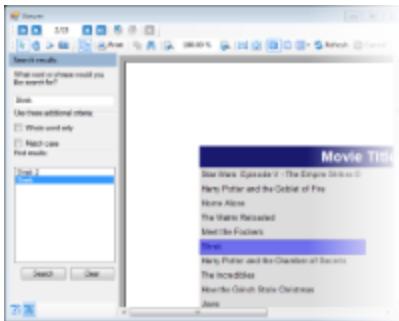
The **Thumbnails** pane appears by default in the sidebar when you click the **Toggle sidebar** button in the toolbar.

This pane comprises 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.



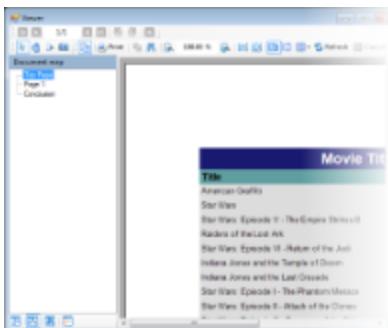
To search in a report:

- 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

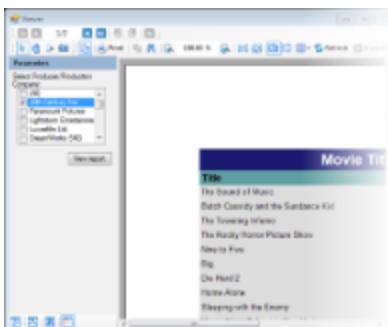
The Documents map pane is enabled for reports where the Label property or the Document map label is set. This pane displays each value for the text box, group, or sub report that you label, and you can click them to navigate to the corresponding area of the report in the Viewer.



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.



- In the **Parameters** pane, you are prompted to enter a value by which to filter the data to display.
- 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.

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 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 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

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 **Use Advanced Printing Options** for further details.

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 **AnnotationToolbarVisible** 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.

ActiveReports and the Web

Professional Edition

With the Professional Edition license, you can use the WebViewer control to quickly display reports in any of four viewer types: **HtmlViewer**, **RawHtml**, **AcrobatReader**, or **FlashViewer**. You can also use the **Silverlight Viewer control** in Silverlight projects.

 **Important:** Before using the WebViewer control, you must first **Configure HTTPHandlers in IIS 7.x**.

In this section

Getting Started with the Web Viewer

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 the Flash Viewer

Learn about the features available with the Flash viewer.

Medium Trust Support

Learn about the features and limitations available in Medium Trust Support environment.

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.

Getting Started with the Web Viewer

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 **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
- **FlashViewer**: Provides an interactive viewing experience and no-touch printing using the widely-adopted Flash Player. See **Using the Flash Viewer** for details.
Client requirements: Adobe Flash Player

To use the WebViewer control

1. In a Visual Studio Web Application, add the WebViewer control to the Visual Studio toolbox. See **Adding ActiveReports Controls** for more information.
2. While in Design view of an ASPX page, from the toolbox, drag the WebViewer control and drop it on the page.
3. With the WebViewer control selected, in the Properties grid, select the ViewerType you want to use. The viewer displays any prerequisites for using the selected ViewerType.
4. To bind a report to the WebViewer, do one of the following:
 - Set the **ReportName** property to the name of a report within your solution.

 **Note:** Alternatively, you can set the **ReportName** property programmatically to a new instance of an ActiveReport class.
For example:
VB code: WebViewer.ReportName="YourReport.rpx"
C# code: WebViewer.ReportName="YourReport.rpx";

- Set the **Report** property to a new instance of an ActiveReport class as shown in the examples below.
To write the code in Visual Basic.NET (Page report)

VB code. Paste INSIDE the Page Load event

```
Dim rpt As New GrapeCity.ActiveReports.PageReport()  
rpt.Load(New System.IO.FileInfo(Server.MapPath("") + "\\invoice.rdlx"))  
WebViewer1.Report = rpt
```

To write the code in C# (Page report)

C# code. Paste INSIDE the Page Load event

```
GrapeCity.ActiveReports.PageReport rpt = new GrapeCity.ActiveReports.PageReport();  
rpt.Load(new System.IO.FileInfo(Server.MapPath("") + "\\invoice.rdlx"));  
WebViewer1.Report = rpt;
```

To write the code in Visual Basic.NET (Section code-based report)

VB code. Paste INSIDE the Page Load event

```
Dim rpt As New MyInvoiceReport()  
WebViewer1.Report = rpt
```

To write the code in C# (Section code-based report)

C# code. Paste INSIDE the Page Load event

```
MyInvoiceReport rpt = new MyInvoiceReport();  
WebViewer1.Report = rpt;
```

To write the code in Visual Basic.NET (Section xml-based report)

VB code. Paste INSIDE the Page Load event

```
Dim sr As New SectionReport()
```

```
sr.LoadLayout(Server.MapPath("") + "\\Invoice.RPX")
WebViewer1.Report = sr
```

To write the code in C# (Section xml-based report)

C# code. Paste INSIDE the Page Load event

```
SectionReport sr = new SectionReport();
sr.LoadLayout(Server.MapPath("") + "\\Invoice.RPX");
WebViewer1.Report = sr;
```

 **Note:** You can use either the **ReportName** property or the **Report** property to bind a report to the WebViewer. If you use both properties, you will get the error message.

5. You must also **Configure HTTPHandlers in IIS 6.x** on your server so that IIS knows how to associate ActiveReports files in the browser.

Using the Flash Viewer

FlashViewer is one of the viewer types of the WebViewer control. It includes Flash representations of the toolbar as well as the sidebar that contains **Table of Contents** and **Thumbnail** tabs.

To use the Flash Viewer, you must copy the following files into your project folder.

- GrapeCity.ActiveReports.Flash.v7.swf
- GrapeCity.ActiveReports.Flash.v7.Resources.swf

 **Note:** GrapeCity.ActiveReports.Flash.v7.Resources.swf is used for localization and is necessary only if you want to use a language resource that is different from the default one. The default locale is U.S. English (en_US).

These files are located in the ...\\ComponentOne\\ActiveReports Developer 7\\Deployment\\Flash folder.

In the WebViewer control **ViewerType** property, when you select **FlashViewer**, you can customize the viewer using the **FlashViewerOptions** properties.

Flash Viewer properties

To access the Flash viewer properties, select the WebViewer on your ASPX page and, in the Properties Window, expand the **FlashViewerOptions** node. If you change the **ViewerType** property to anything other than **FlashViewer**, these property settings are ignored.

Property	Description
DisplayTransparency	Specify whether to print transparent objects.
HyperLinkBackColor	Specify the background color of hyperlinks displayed in the viewer.
HyperLinkForeColor	Specify the color of hyperlink text.
HyperLinkUnderline	Specify whether hyperlink text is underlined.
MultiPageViewColumns	Specify the number of columns to show when the ViewType is set to MultiPage .
MultiPageViewRows	Specify the number of rows to show when the ViewType is set to MultiPage .
PageNumber	Specify the page to display initially.
PrintOptions	Specify how to handle paper orientation during printing. Select from: <ul style="list-style-type: none">• None (orientation is not checked)• Auto (the Flash viewer checks every page, and changes orientation if necessary)
AdjustPaperOrientation	

		<ul style="list-style-type: none">• AdjustByFirstPage (the Flash viewer checks the first page, and if the orientation does not match that of the printer, adjusts the entire report without checking additional pages)
ScalePages		Specify how to handle page scaling during printing. Select from: <ul style="list-style-type: none">• None (pages are not scaled)• Auto (pages are scaled down if they do not fit on the paper)• AllowScaleUp (pages are scaled up or down to best fit the paper)
StartPrint		Specify whether to print the report after loading for one-touch printing. If you set the WebViewer's Height and Width properties to 0, you can print the report without displaying the Print dialog.
ResourceLocale		Specify the Culture for localization. Separate multiple values with commas.
ResourceUrl		Specify a comma-separated list of URLs to SWF files with resource bundles.
SearchResultsBackColor		Specify the background color used to highlight search results text in report pages.
SearchResultsForeColor		Specify the text color for highlighted search results text in report pages.
ShowSplitter		Specify whether to display the splitter, which allows the user to compare report pages in the viewer.
ThemeUrl		Specify the relative URL of a theme to use on the FlashViewer. The following themes are included, and can be found in ...\\ComponentOne\\ActiveReports Developer 7\\Deployment\\Flash\\Themes. Add them to your project to use them. <ul style="list-style-type: none">• FluorescentBlue.swf• Office.swf• OliveGreen.swf• Orange.swf• VistaAero.swf• WindowsClassic.swf• XP.swf
Alignment		Specify the alignment of the table of contents pane. Select from Left or Right.
TocPanelOptions	ShowThumbnails	Specify whether to display a pane with thumbnail views of pages.
	ShowToc	Specify whether to display the table of contents in the FlashViewer.
	Visible	Specify whether to show the table of contents pane initially, without requiring the user to click the Toggle Sidebar button.
	Width	Specify the width of the table of contents pane in pixels.
Url		Specify the relative URL of the FlashViewer control. If you leave this value blank, ActiveReports looks in the main Web folder.
UseClientApi		Specify whether to allow the use of the client API (javascript) for the FlashViewer. If set to False, the Flash viewer ignores any javascript commands sent to it.
ViewType		Specify the page view type. Select from Single, MultiPage, or Continuous.

WindowMode	Specify such display options as transparency, layering, and positioning of the FlashViewer in the browser. Select from:
	<ul style="list-style-type: none"> • Window (displays the Flash viewer in its own rectangular window on the Web page) • Opaque (displays the viewer with a filled background so nothing shows through) • Transparent (displays the viewer with a transparent background so objects in the background show through) This mode may slow animation performance.
Zoom	Specify the zoom level, between 10% and 800%, at which to display the report.

Flash Viewer shortcut keys

You can use the following shortcut keys with the Flash Viewer.

Keyboard Shortcut	Action	Behavior in Internet Explorer
Ctrl + F	Displays the find dialog.	-
F3	Displays the next find result.	Displays the browser's find box.
Esc	Closes the find dialog.	-
Page Down	Moves to the next page.	-
Page Up	Moves to the previous page.	-
Ctrl + P	Displays the print dialog.	Displays the browser's print dialog.
Ctrl + T	Displays the table of contents.	Opens a new tab in the browser.
Ctrl + Home	Moves to the first page.	-
Ctrl + End	Moves to the last page.	-
Ctrl + Right	Moves to the next page.	-
Ctrl + Left	Moves to the previous page.	-
Ctrl + -	Zooms out.	-
Ctrl + +	Zooms in.	-
Left, Right, Up, Down	Moves the visible area of the page in the specified direction.	-
Home, End	Moves to the beginning or end of the current page.	-
Ctrl + 0 (zero)	Sets the zoom ratio to 100%.	-
Ctrl + mouse wheel	Changes the zoom level up or down.	-
Ctrl + M	Displays multiple pages.	Not applicable for IE9 and IE10.
Ctrl + S	Displays a single page.	Displays the Save Webpage dialog in IE9 and IE10.

 **Caution:** As with any other Flash application, **browser** keyboard shortcuts do not work if the Flash Viewer has focus. Click anywhere outside the Flash Viewer to give focus back to the browser to use browser keyboard shortcuts. Likewise, to use the Flash Viewer keyboard shortcuts, click the Flash viewer to give focus back to the Flash Viewer if focus is on the browser.

Also, some shortcut actions are different in Internet Explorer (see **Behavior in Internet Explorer** in the table above), therefore it is recommended to use FireFox instead.

Flash Viewer printing

The Flash Viewer toolbar has a Print button and a Page Range button. Note that you cannot set the page range in the Print dialog, so you must set up a page range prior to printing.

Print

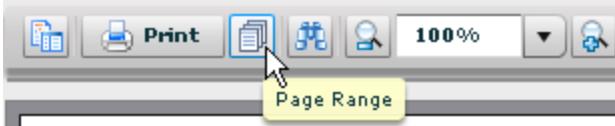
1. On the Flash Viewer toolbar, click the **Print** button.



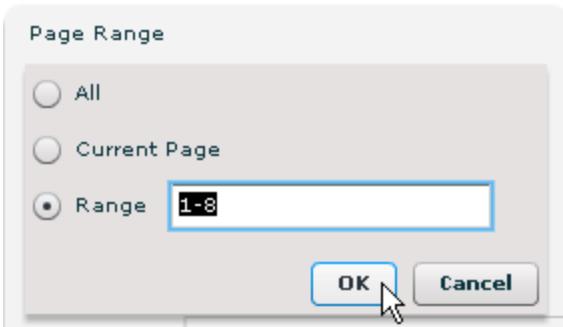
2. In the **Print** dialog that appears, select the printer settings and click **Print**.

Page Range

1. On the Flash Viewer toolbar, click the **Page Range** button.



2. In the **Page range** dialog that appears, select **All** for all pages, **Current Page** for the current page, or **Range** to specify pages for printing and then click **OK**.



3. On the Flash Viewer toolbar, click the **Print** button and then, in the **Print** dialog that appears, click **Print**.

Using the HTML Viewer

HtmlViewer 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**, **Table of Contents** and **Search** panes.

HTML Viewer Properties

These **HtmlExportOptions** properties on the WebViewer control apply only when you select the **HTML ViewerType**. If you change the ViewerType to another value, these settings are ignored.

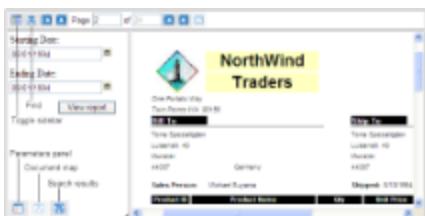
Property

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 HtmlViewer downloads only HTML and javascript to the client browser.



HTML Viewer Toolbar

The HTML viewer toolbar offers various ways to navigate through reports.

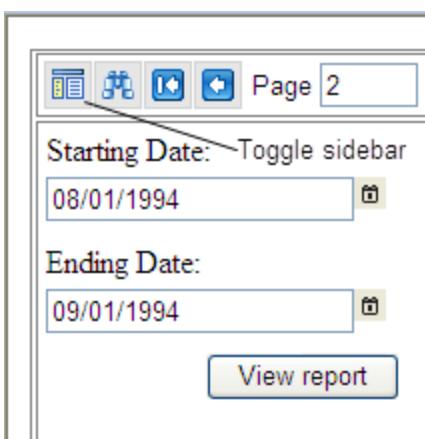
Toolbar Element	Name	Description
	Toggle Sidebar	Displays the sidebar that includes the Parameters, Table of Contents and Search panes.
	Find	Displays the Search pane of the sidebar.
Page <input type="text" value="2"/> of <input type="text" value="8"/>	Go to page	Opens a specific page in the report. To view a specific page, type the page number and press ENTER.
	Go to Previous/Next page	Navigates through a report page by page.
	Go to First/Last page	Jumps to the first or last page of a report.
	Back to parent report	Returns to the parent report in a drill-down page 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 **View report**.



If a report does not have parameters, the Parameters pane of the sidebar is disabled.

HTML Viewer Table of Contents

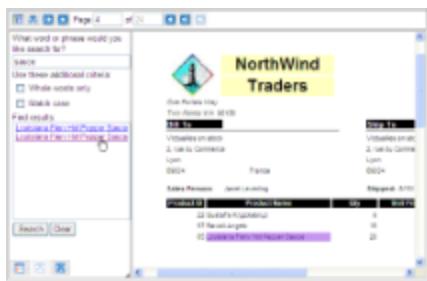
To display the Table of Contents pane, in the toolbar, click **Toggle Sidebar**. Then at the bottom of the sidebar, click the **Table of Contents** button.

Note that the Table of Contents pane is only enabled 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.

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.

To start a new search, click **Clear** under the **Find results** list.



Working with HTML Viewer using Javascript

In order to work with HTML viewer using the Javascript, you need to follow a set of procedures. This involves calling the **Loaded** event first and then obtaining an instance of the **ViewerViewModel**. Once you have access to the **ViewerViewModel**, use its API methods and properties to work with the HTML Viewer.

Loaded Event

HtmlViewer raises a **Loaded** event to notify the listeners that the internal initialization is complete. Following is the sample code to raise a **Loaded** event:

```
$(document).ready(function () {
  $('#' + clientId).bind('loaded', function() {
    ...
  });
});
```

Note: You can obtain the **ClientId** from the **WebViewer** control.

ViewerViewModel

In order to work with the API, obtain an instance of the **ViewerViewModel** first, using the **GetViewModel(clientId)** javascript function that searches **ViewerViewModel** array through the **clientId**. This function throws an exception if there is no **ViewerViewModel** with the requested **clientId** available.

Use code like the following to call an instance of the **ViewerViewModel**:

```
var viewModel = GetViewModel(clientId);
```

After obtaining the **ViewerViewModel**, the code can call API methods and properties.

Methods/Properties	Example	Description
Sidebar	this.Sidebar	Gets the Sidebar view model instance.
Toolbar	this.Toolbar	Gets the Toolbar view model instance.
PageLoaded	this.PageLoaded(false);	Gets the Boolean value indicating whether the page was loaded or not.
Export	this.Export(exportType, callback, saveAsDialog, settings)	Exports the loaded page to a specified format. In order to export without any errors the PageLoaded() property must be True. <ul style="list-style-type: none"> • exportType: Requested output format • callback: Function which obtains the URI of the exported document. • saveAsDialog: Optional request to show save as dialog after export. • settings: Optional export settings.

Note: ExportType is an Enumeration.

```
var ExportType = { Pdf, Html, Word, Xls, Xml };
```

Print	<code>this.Print()</code>	Prints the report using pdf printing. In order to print without any errors the PageLoaded property must be True.
SidebarViewModel		
		SidebarViewModel allows you access to its various properties and methods to get the current state and show/hide sidebar and sidebar panels.
Methods/Properties	Example	Description
IsSidebarVisible	<code>this.IsSidebarVisible(false);</code>	Gets the Boolean value for entire sidebar visibility.
HideShowSidebar	<code>this.HideShowSidebar()</code>	Toggles Sidebar's visibility.
IsBookmarksPaneVisible	<code>this.IsBookmarksPaneVisible(false);</code>	Gets the Bookmarks Pane's visibility.
ShowBookmarksPane	<code>this.ShowBookmarksPane()</code>	Toggles Bookmarks Pane's visibility.
IsParametersPaneVisible	<code>this.IsParametersPaneVisible(false);</code>	Gets the Parameters Pane's visibility.
ShowParametersPane	<code>this.ShowParametersPane()</code>	Toggles Parameters Pane's visibility.
IsSearchPaneVisible	<code>this.IsSearchPaneVisible(false);</code>	Gets the Search Pane's visibility.
ShowSearchPane	<code>this.ShowSearchPane()</code>	Toggles Search Pane's visibility.
HideAll	<code>this.HideAll()</code>	Hides all Sidebar panes.
ToolbarViewModel		
		A ToolbarViewModel provides access to various toolbar properties and methods to get its current state and work with loaded reports.
Methods/Properties	Example	Description
Enabled	<code>this.Enabled(false);</code>	Gets the Boolean value determining whether the report was loaded successfully and the user can change the page or not.
PageCount	<code>this.PageCount(0);</code>	Gets the page count of the loaded report.
CurrentPage	<code>this.CurrentPage(0);</code>	Gets the currently opened page number.
GoToPage	<code>this.GoToPage(number, force, callback)</code>	Opens the specified page of the loaded report.
 Note: The Current Page starts from 1 after the report is loaded.		

Medium Trust Support

All features of ActiveReports Developer are available without restrictions in a Full trust environment. You can also use ActiveReports Developer under Medium trust, but with limitations on some of the features.

 **Caution:** 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.

 **Note:** If you see an evaluation banner when deploying your ActiveReports Developer project, you should use the **Web Key Generator** utility to create the Web Key and integrate the license information into your project.

Feature Limitations

- Exporting**
 - **RTF, Text, TIFF** and **Excel** filters are not supported in Medium trust.
 - For the **PDF export filter** and for the **PDF rendering extension**, digital signatures are not supported.
- The **End User Designer** and Windows Form Viewer controls require Full trust.
- The **Picture** control does not support **metafiles**, which require Full trust.
- The **ImageType** property of the **Chart** control must be set to **PNG**.
- OleObject** and Custom controls require Full trust.
- Scripting** requires Full trust, so if you need to use code in reports under Medium trust, use code-based reports rather than RPX format.

Recommended Development Environment for Medium Trust Tests

To set up a Medium trust environment

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 (located in the \\Windows\\Microsoft.NET\\Framework\\v4.0.30319\\Config folder).

Using the Silverlight Viewer

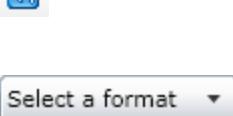
ActiveReports provides a Silverlight Viewer to where you can load and view your reports. This viewer contains a toolbar and a sidebar with **Search**, **Table of Contents** and **Parameters** panes.

 **Note:** Microsoft Silverlight 4 Tools are required for application development with the ActiveReports Silverlight Viewer.

Silverlight Viewer toolbar



Toolbar element	Name	Description
	Toggle Sidebar	Displays the sidebar that includes the Search , TOC (Table of Contents) , and Parameters panes.
	Print	Displays the Print dialog where you can specify the printing options.
	Find	Displays the Search pane in the sidebar.
	Zoom out	Decreases the magnification of your report.
	Zoom reset	Resets the magnification to default
	Zoom slider	Allows you to drag the slider to increase or decrease the magnification of your report.
	Zoom in	Increases the magnification of your report.
	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.
 8/26	Current page	Shows the current page number and 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 to the parent report in a drillthrough page report.
	Export Format	Allows you to specify the export format for a report. You can select from the following options: PDF, Excel, HTML, Word, or XML.
	Export Report	Exports a report into the selected format.

Silverlight Viewer Sidebar

Search Pane

The Search pane appears by default in the sidebar when you click the Toggle Sidebar button. This pane lets you enter a word or phrase to search within the report.

To search in a report:

1. Enter the word or phrase in the search field.
2. Under **Use this 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.
3. Click the **Search** button to see the results appear in the **Find results** list.
4. 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.

Table of Contents (TOC) Pane

To display the **Table of Contents** pane, in the toolbar, click **Toggle Sidebar**. Then at the bottom of the sidebar, click the **Table of Contents** button.

If a report does not have the Label property or Document map label set, the Table of Contents (TOC) pane does not appear in the sidebar.

Parameters Pane

The Silverlight Viewer allows you to view reports with parameters. In the toolbar, click the **Toggle sidebar** button to open the 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.

Display the report in the Viewer

Set up your Silverlight project using the following steps: (see the Silverlight walkthrough for more details)

1. Create a new Silverlight project or open an existing one, ensuring that the Silverlight Version option is set to **Silverlight 4** or higher.
2. In the Visual Studio Solution Explorer, right-click *YourProject.Web* and select **Add**, then **New Item**.
3. In the Add New Item dialog that appears, select the Reporting template, then select **ActiveReports 7 Web Service**. This adds ActiveReports.ReportService1.asmx to your project.
4. From the Toolbox ActiveReports 7 tab, drag the **Viewer** control and drop it on the design view of *MainPage.xaml*.
5. In the Solution Explorer, right-click *YourProject.Web* and select **Add**, then **Existing Item** and select an existing report to load in the viewer.
6. On *MainPage.xaml*, with the viewer selected, go to the Properties window and double click the Loaded event.
7. In the *MainPage* 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 a .rpx report as well.

Visual Basic.NET code. Paste INSIDE the *viewer1_Loaded* event in *MainPage.xaml.vb*.

```
Viewer1.LoadFromService("YourReportName.rdlx")
```

C# code. Paste INSIDE the *viewer1_Loaded* event in *MainPage.xaml.cs*.

```
viewer1.LoadFromService("YourReportName.rdlx");
```

To avoid evaluation banners appearing at runtime, license your ActiveReports Silverlight project. You can find information on licensing ActiveReports Silverlight in **License Your ActiveReports** under To license an ActiveReports Silverlight project.

Silverlight Viewer printing

1. In the Silverlight viewer toolbar, click the **Print** button.
2. In the **Print** dialog that appears, select the printer settings and click **Print**.

In addition to the standard Silverlight print option, you can setup PDF printing in your Silverlight project and print a document from Silverlight to the PDF format directly.

 **Note:** PDF printing is not supported in the Silverlight Out-of-Browser applications.

See **Provide PDF Printing in the Silverlight Viewer (Pro Edition)** to learn how you can set up and print a report from the Silverlight Viewer to PDF format.

Silverlight feature limitations

- Limitations on printing:
 - ActiveReports Silverlight Viewer does not provide the one-touch printing option.
 - The maximum value for the print range is 2000 pages.
- Vertical text is not supported in the ActiveReports Silverlight Viewer.
- Horizontal and vertical text of MS PMincho and some other ideographic characters may render incorrectly in the ActiveReports Silverlight Viewer.
- The use of the Silverlight Viewer control in layout panels has a limitation related to the default size of the panel.
- The multipage mode is not available in the ActiveReports Silverlight Viewer.
- To display a report with annotations in the Thumbnails view correctly, hide the Thumbnails view by clicking the

Show TOC/Thumbnails icon in the Toolbar and then open the report in the Silverlight Viewer.

- Some Silverlight properties and events are not supported in XAML. However, you can still use these properties by setting them in code as follows:

Visual Basic.NET code. Add this code in an event like Button_Click.

```
Viewer1.FontStyle = FontStyle.Italic
```

C# code. Add this code in an event like Button_Click.

```
viewer1.FontStyle = FontStyle.Italic;
```

Silverlight properties and events that are not supported

Properties

- FontStyle
- FontWeight
- FontFamily
- FontSize
- Foreground
- FontStretch
- Padding
- VerticalContentAlignment
- HorizontalContentAlignment
- IsTabStop

Events

- GotFocus
- LostFocus

Using the 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.

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.
	Copy	Copies text that you select in the Selection mode to the clipboard.

 **Note:** In case the GrapeCity.ActiveReports.Export.Xml.v7.dll and GrapeCity.ActiveReports.Export.Word.v7.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.
100.00 %	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.
	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.
1/13	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.
	Refresh	Refreshes the report.
	Cancel	Cancels the report rendering.
	Pan mode	A hand serves as the pointer that you can use to navigate the report.
	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.

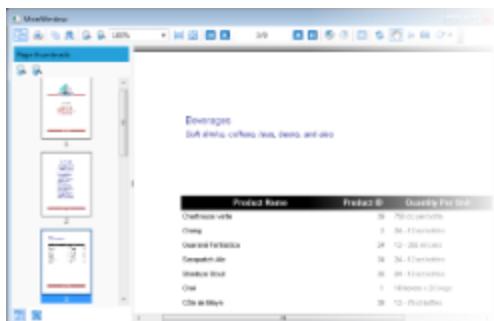
WPF Viewer Sidebar

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

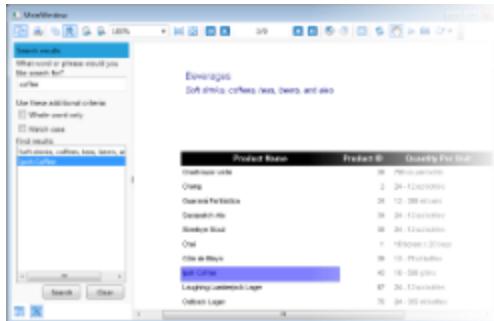
The **Thumbnails** pane appears by default in the sidebar when you click the **Toggle sidebar** button in the toolbar.

This pane comprises 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.



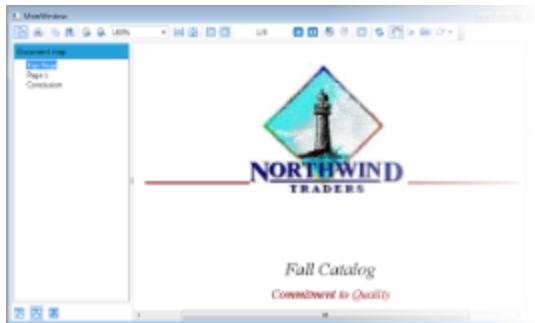
To search in a report:

- 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

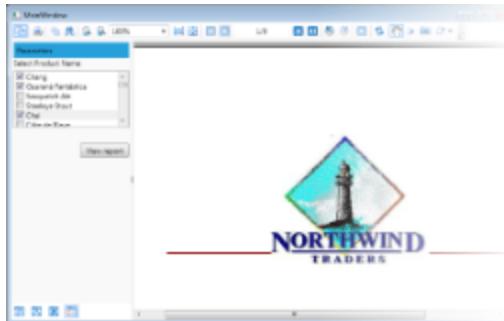
The Documents map pane is enabled for reports where the Label property or the Document map label is set. This pane displays each value for the text box, group, or sub report that you label, and you can click them to navigate to the corresponding area of the report in the Viewer.



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 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.

Display the report in the WPF Viewer

Set up your WPF Application project by using the following steps.

1. Create a new WPF Application project or open an existing one.
2. For a new project, in the Visual Studio Solution Explorer, right-click *YourProject* and select **Add**, then **New Item**.
3. In the Add New Item dialog that appears, select the **ActiveReports 7 Page Report**, **ActiveReports 7 Section Report (code-based)** or **ActiveReports 7 Section Report (xml-based)**. This adds the necessary references to your project.
4. From the Toolbox ActiveReports 7 tab, drag the **Viewer** control and drop it on the design view of *MainWindow.xaml*.
5. In the Solution Explorer, right-click *YourProject* and select **Add**, then **Existing Item** and select an existing report to load in the viewer.
6. In the Properties window, with the report selected, set **Copy to Output Directory** to **Copy Always**.
7. On *MainWindow.xaml*, with the viewer selected, go to the Properties window and double click the Loaded event.
8. In the *MainWindow* code view that appears, add code like the following to the *viewer1_Loaded* event to bind the report to 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).

Note: Refer to **LoadDocument** ('**LoadDocument Method**' in the on-line documentation)

method to see other ways to load a report in WPF Viewer.

To write the code in Visual Basic.NET

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

Visual Basic.NET code. Paste INSIDE the viewer1_Loaded event in MainWindow.xaml.vb.

```
Viewer1.LoadDocument("YourReportName.rdlx")
```

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

Visual Basic.NET code. Paste INSIDE the viewer1_Loaded event in MainWindow.xaml.vb.

```
viewer1.LoadDocument(new YourReportName())
```

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

Visual Basic.NET code. Paste INSIDE the viewer1_Loaded event in MainWindow.xaml.vb.

```
Viewer1.LoadDocument("YourReportName.rpx")
```

To write the code in C#

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

C# code. Paste INSIDE the viewer1_Loaded event in MainWindow.xaml.cs.

```
viewer1.LoadDocument("YourReportName.rdlx");
```

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

C# code. Paste INSIDE the viewer1_Loaded event in MainWindow.xaml.cs.

```
viewer1.LoadDocument(new YourReportName());
```

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

C# code. Paste INSIDE the viewer1_Loaded event in MainWindow.xaml.cs.

```
viewer1.LoadDocument("YourReportName.rpx");
```

Additional Features

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

Annotations Toolbar

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



Annotations are added via the WPF 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

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 WPF Viewer.

Keyboard Shortcut

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.
F5	Refreshes the report.

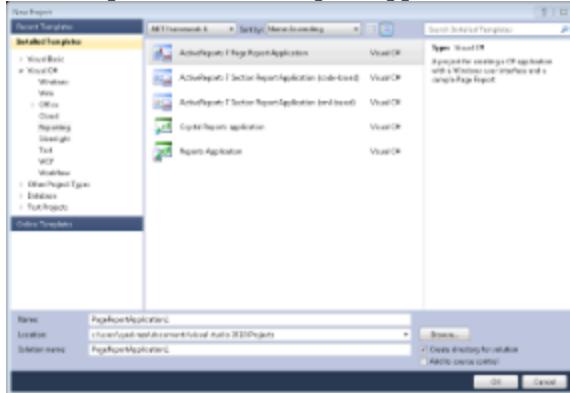
Adding an ActiveReports Application

ActiveReports Developer provides an in-built sample application that includes a report template along with the Viewer

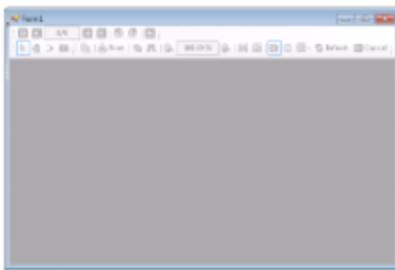
control. You learnt about creating a report and viewing it in the preceding topics. See **Adding an ActiveReport to a Project** and **Viewing Reports** for further details. With this Windows Forms application you only need to create a report layout and run the application to view the report, effectively skipping the manual process of adding a Viewer and template separately and creating an instance of the report.

To add an ActiveReports application to a project

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 7 Page Report Application
 - ActiveReports 7 Section Report Application (code-based)
 - ActiveReports 7 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. 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.



 **Note:** If you run the application right after the new report project is created, a blank report appears on the Form.

Concepts

Learn about concepts that help you to understand how best to use ActiveReports.

This section contains information about

ActiveReports Designer

Learn what each of the tools and UI items on the report designer can help you to accomplish.

Report Types

Learn which type of ActiveReport best suits your needs.

Page Report Concepts

Learn basic concepts that apply to Page reports.

Section Report Concepts

Learn basic concepts that apply to Section reports.

Text Justification

Learn how to use the TextJustify property.

Multiline in Report Controls

Learn how to enter multiline text in a report.

Line Spacing and Character Spacing

Learn how to control line spacing and character spacing in TextBox and Label report controls.

Exporting

Learn about rendering extensions and exports, and which formats are available with Page and Section reports.

Interactive Features

Learn about various interactive features that affect your report output.

Windows Forms Viewer Customization

Learn about all of the ways that you can customize the Windows Forms Viewer control.

Designer Control (Pro Edition)

Learn how you can provide a Windows Forms report designer for your end users.

Shrink Text to Fit in a Control

Learn how to shrink text to fit in a TextBox control.

Standalone Designer and Viewer

Learn about the Standalone Designer and Viewer applications that help you create, edit and view a report quickly.

Localization

Learn about the ActiveReports localization model.

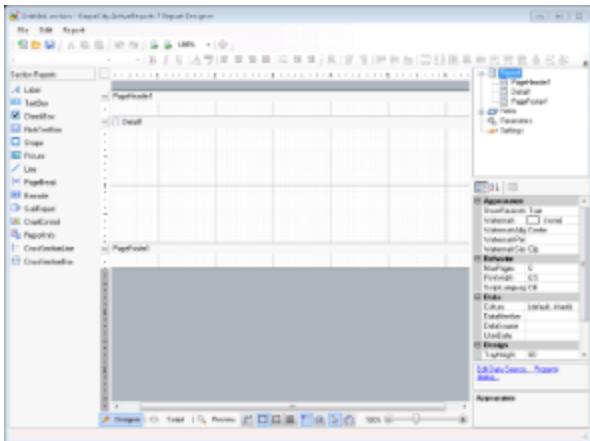
ActiveReports Designer

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 two types of report layouts: section layout and page layout.

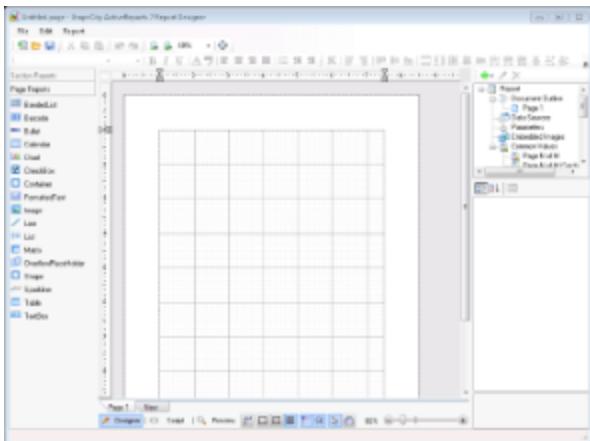
Section Report Layout

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.



In this section

Design View

Explore the elements of the design tab that appear with both types of reports.

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 FPL 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

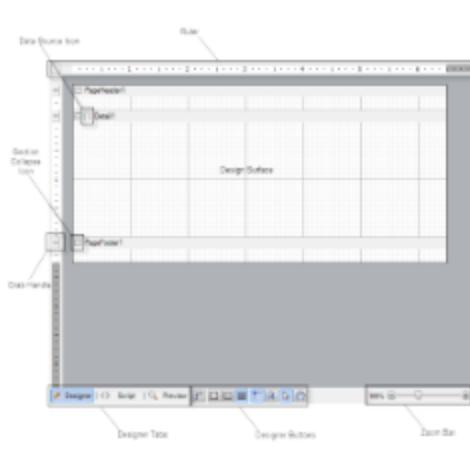
Get the ability to zoom in or zoom out of your report layout.

Design View

The report designer is fully integrated with the Microsoft VisualStudio IDE. In this topic, we introduce the main parts of the designer in section and page reports 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

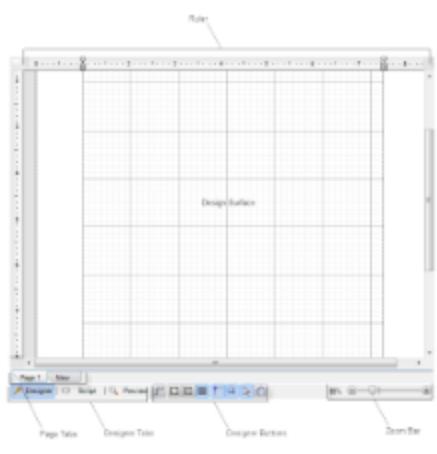
Designer buttons are located below the design surface next to the designer tabs. **Dimension Lines**, **Hide Grid**, **Dots**, **Lines**, **Snap to Lines**, and **Snap to Grid** buttons help you to align report controls and data regions. The **Select Mode** and **Pan Mode** buttons determine whether 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

In a page 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

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.

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.
View:	Opens the Designer, Script or Preview tabs respectively. See Designer Tabs for more details. This menu item appears only with XML based section reports.
• Designer	
• Script	
• Preview	

Report menu for page reports

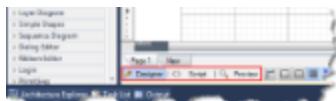
Menu Item	Description
Save	Opens the Save As dialog to save the newly created report in RDLX file format.

Layout

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 CPL Report	Converts an FPL report to a CPL report. This menu item is enabled when you have an FPL report open.
Convert to FPL Report	Converts a CPL report to an FPL report. This menu item is enabled when you have a CPL report open.
Convert to Master Report	Converts a CPL report to a Master Report. This menu item is enabled when you have a CPL report open. It disappears from the Report menu when a master report is applied to the report through Set Master Report . See Master Reports 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.
Set Master Report	Opens the Open dialog to select a Master Report (RDLX-master file format) to apply to the current project. This menu item is enabled when you have a CPL report open. It is disabled when an FPL report is open or when a Master Report is created with the Convert to Master Report menu item.
Generate Excel Template	Creates an Excel template of the report that you or the end user can customize for use with Excel exports to control the way the exported reports appear. This menu item is enabled when you have a CPL report open.
View	Opens the Designer, Script or Preview tab. See Designer Tabs for more details.
• Designer	
• Script	
• Preview	
Page Header	Toggles the report Page Header on or off. This menu item is enabled when you have a CPL report open.
Page Footer	Toggles the report Page Footer on or off. This menu item is enabled when you have a CPL report open.

Designer Tabs

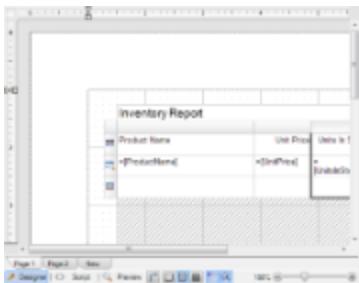
The Designer has three tabs located at the bottom of the report design surface. Create a report layout in the Designer tab, write a script in the Script tab to implement .NET functionality and see the result in the Preview tab.



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.

The generated reports serve as standalone 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, you can embed code blocks that you can reference in the expressions you use on report controls. See [Using Script in a Page Report](#) for more information about using script in page reports.

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

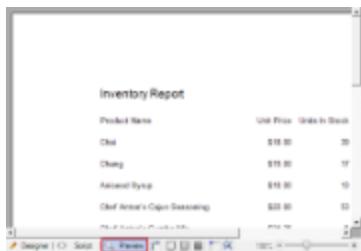
The **Preview tab** allows you to view your report without the need to actually run your project. This makes it easy to quickly see the run-time impact of changes you make in the designer or the code.

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 in Visual Studio 2008 / 2010, the **Preview tab** is not visible and thus you cannot preview.



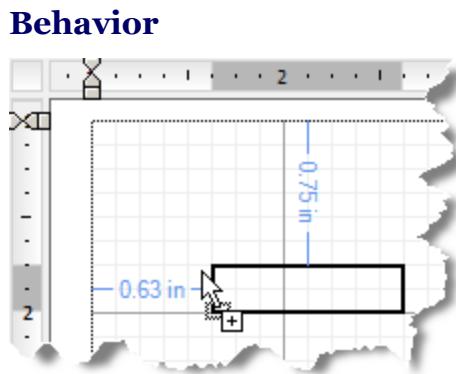
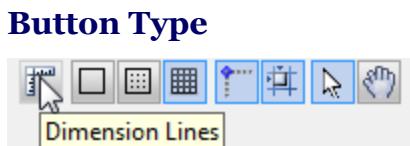
Designer Buttons

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.

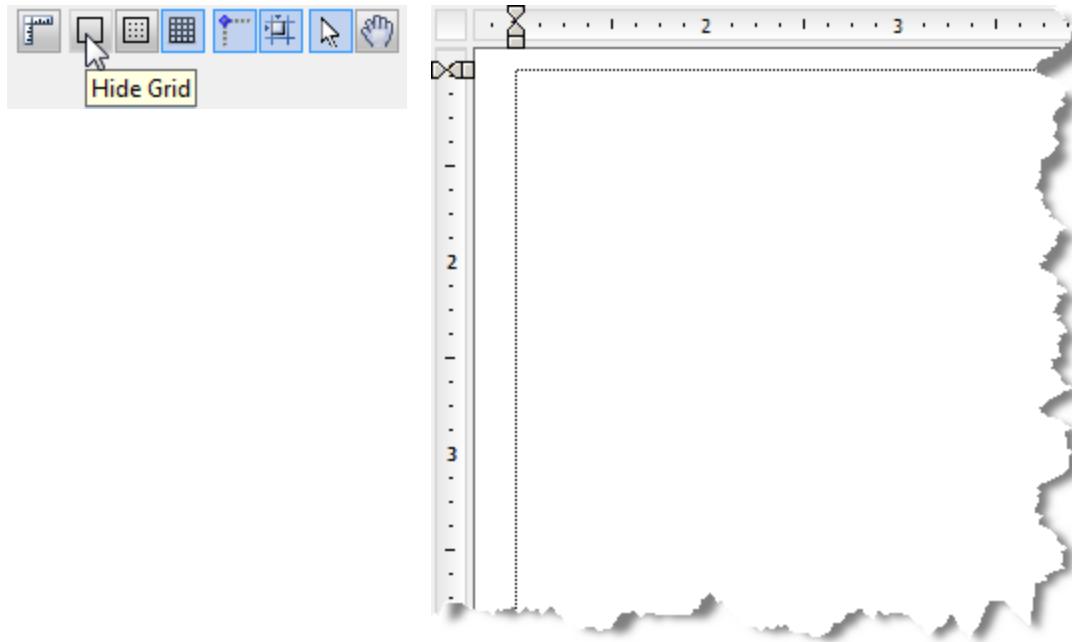


Note: With section reports, you can change the number of grid columns and rows in the Report Settings dialog on the Global Settings tab. With page reports, you can change the grid spacing in the Report Properties dialog on the General tab.

Hide Grid

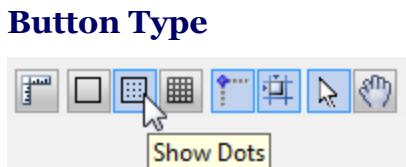
By default, grid lines and dots appear on the report design surface. You can click this button to hide the grid and design your report on a blank page. Lines or dots are also removed from the design surface when you hide the grid, but Snap to Lines or Snap to Grid settings remain unaffected.



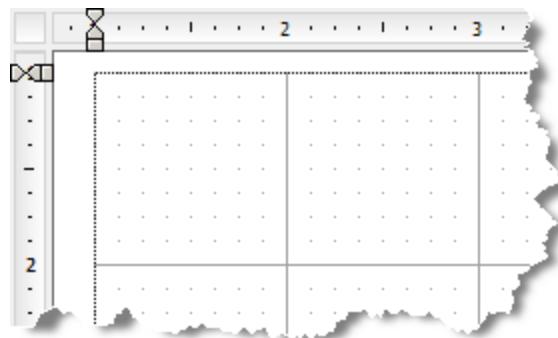


Show Dots

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.



Behavior

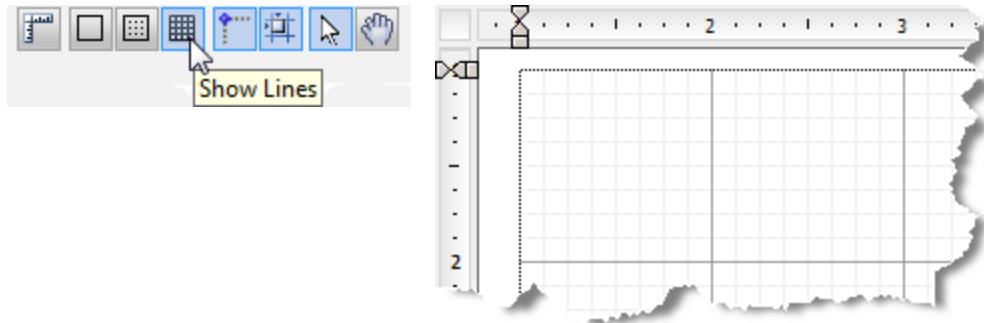


Show Lines

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.



Behavior



Note: Only one option out of Hide Grid, Show Dots and Show Lines can be selected at one time.

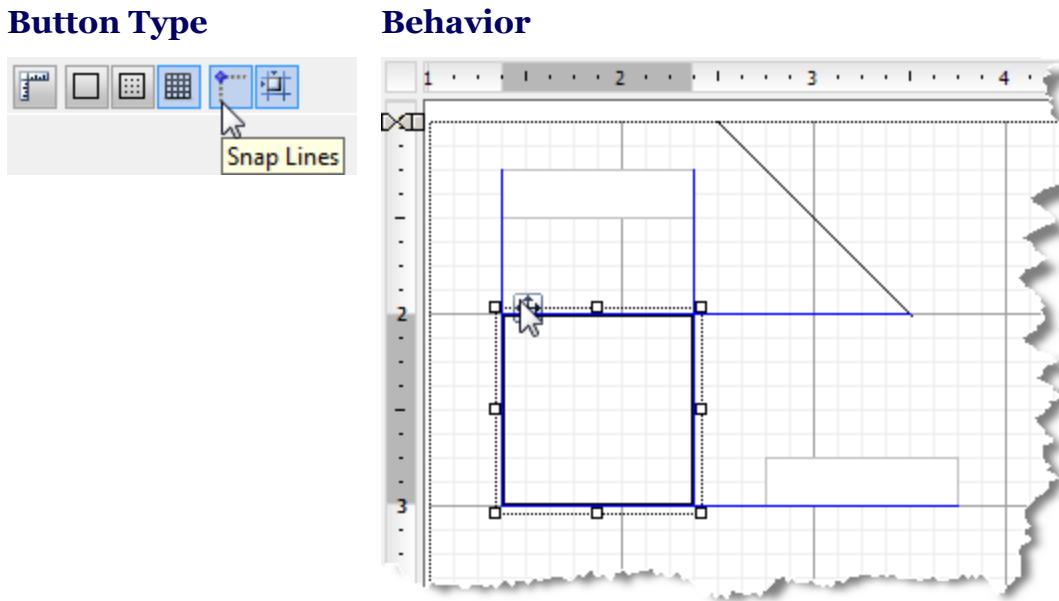
Control Drag and Drop Settings

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

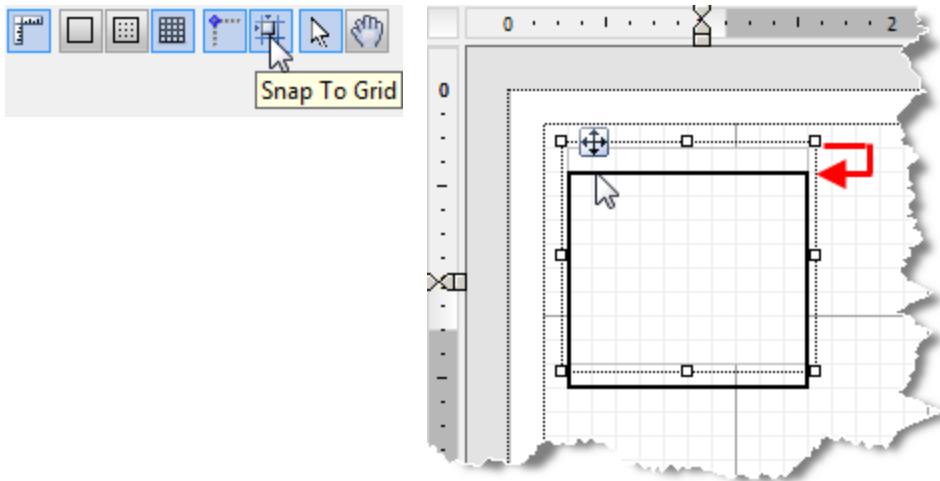
This setting aligns the control you are dragging with other controls on the report design surface. When you drag the control around, snap lines appear when it is aligned with other controls or with the edges of the report, and when you drop it, it snaps into place in perfect alignment. See **Snap Lines** for more information.



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.





Mouse Modes

These settings allow you to specify how you want the mouse to behave in the designer.

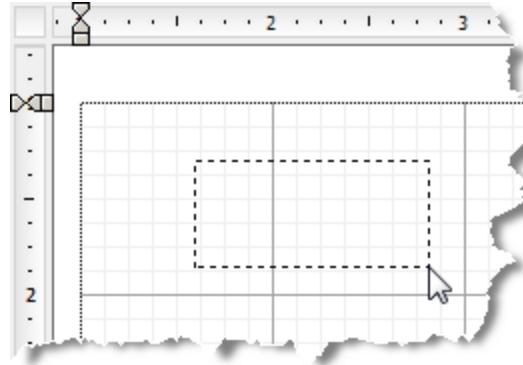
Select Mode

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

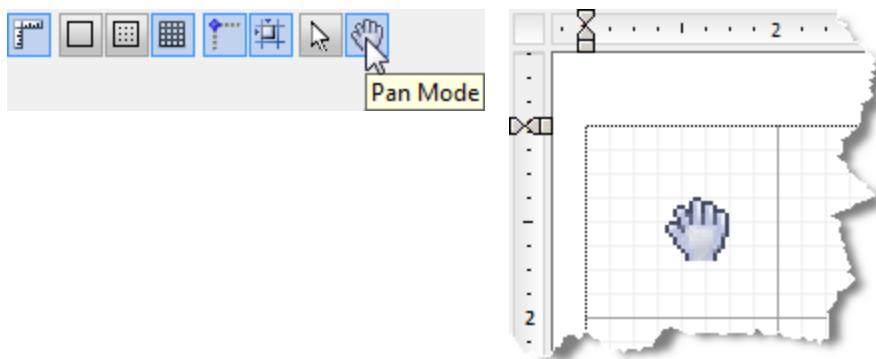
Use the Pan mode to make navigation easier. In this mode, you cannot select, edit, add or delete a control from the design surface. A hand cursor appears in Pan mode and you can navigate through your report by pressing the left mouse button and dragging the report to the desired position.



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

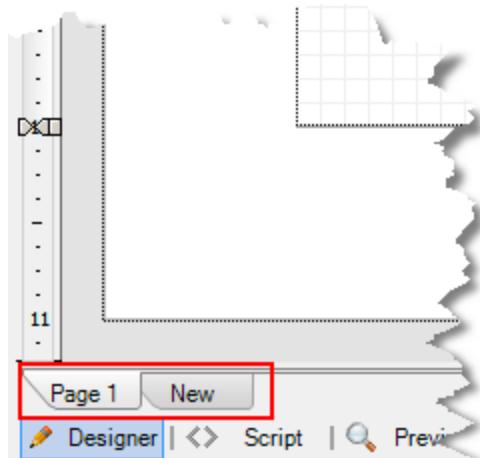
Behavior



Page Tabs

Page tabs appear in an Excel-like bar below the report design surface. This feature is only available in FPL 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 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 comprises 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 Developer**.

The **ActiveReports** toolbar appears under the Visual Studio menu bar. The toolbar options may differ based on whether you have a section report or a page 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.

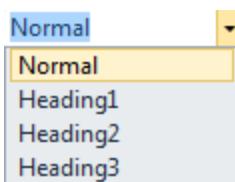
Text Decoration

Command

Description

Style

Offers a dropdown list of selectable styles for formatting text in controls like Label, TextBox, CheckBox and ReportInfo. These styles are available in a section report only.



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, in a data region like Matrix 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, for a data region like Matrix 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 a section report, for the RichTextBox control, bold applies to the selected text only. In a page report, for a data region like Matrix 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, for a data region like Matrix 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, for a data region like Matrix 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

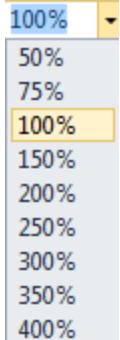
Zoom Out Reduces the magnification level of the design surface and any elements within it.



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 Horizontal Spacing Decreases the horizontal spacing by one grid unit with respect to the primary control.



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.
Indent	Increases the indent of selected text in the RichTextBox control area in a section report.
Outdent	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.

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 two 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)

Section 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 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 Developer** 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 7**.

To change a report using the Report Explorer

More actions specific to each report type can be found in **Exploring Page 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 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 `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**.

Exploring Page Reports

Page reports can be of two types - an **FPL**, fixed page layout report, and a **CPL**, continuous page layout report. You can convert FPL reports to CPL and CPL reports to FPL - see **Report Menu** for details. FPL and CPL reports have some differences in report elements, so a Report Explorer shows different nodes for an FPL and a CPL.

When you have a page report open in the **ActiveReports Designer**, an FPL type is displayed by default and you can see nodes like the following.

- Document Outline
 - Each report page
 - Each control on the page, for example:
 - BandedList
 - Matrix
 - 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)
 - EmbeddedImages (right-click to browse for an image to add)
 - CommonValues (drag onto the report to display the value in a textbox)

To have a CPL open in the **ActiveReports Designer**, you should open a page report - an FPL by default, and then convert it to a CPL.

To convert an FPL to a CPL

1. In the **File** menu, select **Open** to open an existing page report in the **ActiveReports Designer**, or select **New...** and then **Page Report** to create a new page report.
2. In the **Report** menu, click **Convert to CPL Report**.

After you have an FPL converted to a CPL, you can see nodes like the following:

- Document Outline
- Body (default, cannot be removed)
- Page Header and Footer (can be added)
 - Each control, for example:
 - TextBox
 - Picture
 - PageBreak
 - SubReport
- 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)
 - EmbeddedImages (right-click to browse for an image to add)
 - CommonValues (drag onto the report to display the value in a textbox)

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 share a data source; add datasets; add, edit, or remove embedded images; and drag common values like page numbers, current date, or report name onto the report as a textbox. 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. 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, C:\Program Files (x86)\ComponentOne\ActiveReports 6\Data\NWIND.MDB.
5. Under **Log on to server**, select the radio button next to **Use Windows NT integrated security**.
6. 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.

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 7 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 7 Page Report** toolbox becomes available. For information about the report controls available in this toolbox, please see the **Page Report Toolbox** topic.

When a Windows Form has focus, the **ActiveReports 7** 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 7** toolbox group offers one Web control: the WebViewer (requires Professional Edition license). For more information, see **Getting Started with the Web Viewer**.

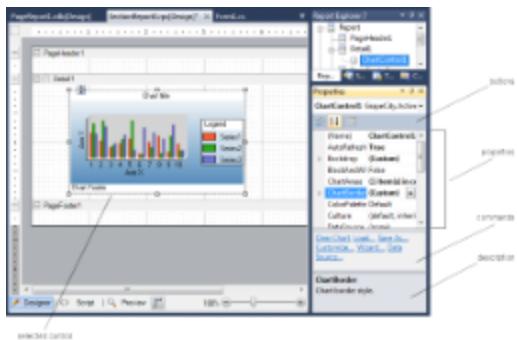
Properties Window

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.

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 section reports, you can change the units of measure for the rulers. See **Change Ruler Measurements** for further details.

In **Page 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.



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 Continuous Page Layout (CPL), when the mouse button is released during auto scrolling at a location outside the design surface, the surface extends to accommodate the control.

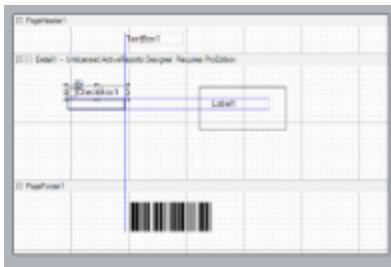
Scrolling stops in the following scenarios:

- 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.

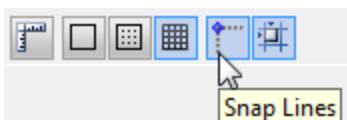
 **Tip:** To enable auto scrolling for multiple controls, hold down the **[Ctrl]** or **[Shift]** key to select the controls. Drag them together to the edge of the design surface and enable auto scrolling.

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 Standalone 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

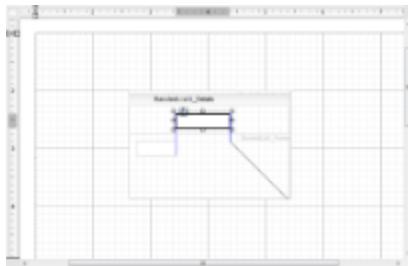
On dragging with a mouse

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.



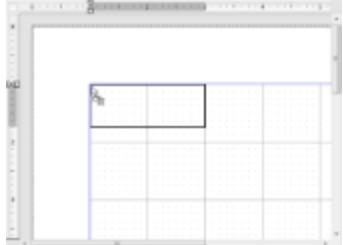
- Another control inside the same data region.



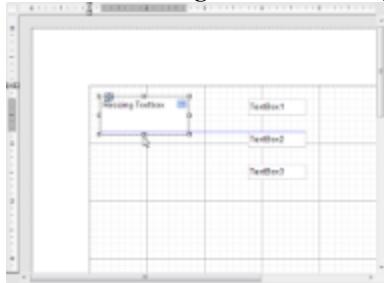
- Parts of a data region (bands in a **BandedList**, or columns and rows in a **Table**).



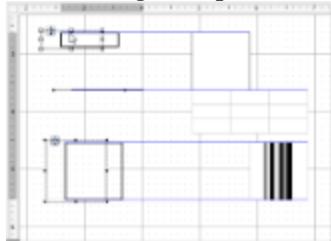
- Report edges and section edges.



- 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] + o** : Return to 100%

Report Types

ActiveReports provides a number of ways to design a report. You can choose 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.

In this section

CPL Page Report

FPL Page Report

Code-Based Section Report

XML-Based Section Report

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.

This layout has two variations, **Continuous Page Layout (CPL)** and **Fixed Page Layout (FPL)**. You can convert between these formats using the Visual Studio **Report** menu that appears when a Page Report has focus. The difference between these two layouts is in the way that they handle data. In a CPL report, controls grow vertically to accommodate data. In an FPL report, controls do not change in size based on the data, but you can use an **OverflowPlaceholder** to handle any extra data.

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 **Adding an ActiveReport to a Project** for details on how to access report templates.

- **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 Report Definition Language.
- **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. See **Master Reports** for further details.
- **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	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 (FPL)	Page report (CPL)
Viewers & Editors			
Visual Studio Integrated Designer	✓	✓	✓
Expressions Editor	✗	✓	✓
Designer Script Editor	✓	✓	✓
Windows Form Viewer	✓	✓	✓
WebViewer (Pro Edition). Includes viewer types HTML, RawHTML, PDF and Flash.	✓	✓	✓
HTTP Handlers (Pro Edition)	✓	✓	✓
Silverlight Viewer (Pro Edition)	✓	✓	✓
Report Controls			
BandedList	✗	✓	✓
List	✗	✓	✓
Matrix	✗	✓	✓
Table	✗	✓	✓
OverflowPlaceHolder	✗	✓	✗
Chart	✓	✓	✓
Barcode	✓	✓	✓
Bullet	✗	✓	✓
Calendar	✗	✓	✓
CheckBox	✓	✓	✓
Container	✗	✓	✓
CrossSectionLine	✓	✗	✗
CrossSectionBox	✓	✗	✗
FormattedText	✗	✓	✓
Image	✗	✓	✓
Label	✓	✗	✗
Line	✓	✓	✓
OleObject	✓	✗	✗
Pagebreak	✓	✗	✗
Picture	✓	✗	✗
ReportInfo	✓	✗	✗
RichTextBox	✓	✗	✗

Shape	✓	✓	✓
Sparkline	✗	✓	✓
Subreport	✓	✗	✓
TextBox	✓	✓	✓

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	✓	✓	✓

Standalone Applications

ActiveReports Developer Viewer	✓	✓	✓
ActiveReports Developer Theme Editor	✗	✓	✓
ActiveReports Developer Designer (standalone application)	✓	✓	✓

CPL Page Report

The Continuous Page Layout (CPL) 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 Page report to a project, it is an FPL report by default. To change to a CPL report, drop down the

Report menu and select **Convert to CPL report**. The OverflowPlaceholder control disappears from the toolbox, and the page tabs disappear from below the report design surface.

Master Reports

One way in which CPL reports differ from FPL 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**.

Themes

Both FPL and CPL 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

CPL 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

Page 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. For more information, see **Shared Data Source (RDSX)**.

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 page 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 Calendar report control, and List, Table, and Matrix 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 **Page Report Toolbox**.

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 Matrix 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 in a Page report**.

Drill Down

You can use the Visibility settings available on report controls, data regions, table rows, and matrix 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 CPL reports using PageSize settings, as well as PageBreakBefore and PageBreakAfter properties on data regions, groups, and rectangles.

FPL Page Report

The new Fixed Page Layout (FPL) 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 FPL reports, there is no need to use code or add measurements to make sure that everything fits. Unlike the CPL Page 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**.

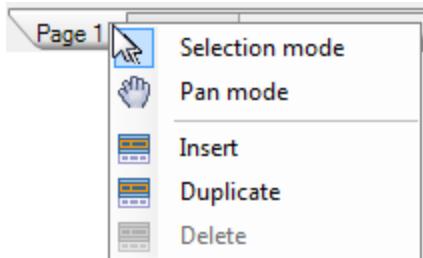
Where does the rest of the data go?

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 **Set Up Collation** and **Collate Multiple Copies of a Report**.

Caution: Fixed Page Layout (FPL) 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.

Code-Based Section Report

When you add an ActiveReports 7 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** 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 **Work with Data in Section Reports** topic.

Viewing and Exporting

To display a code-based report in the viewer, you use the LoadDocument method of the viewer. See **Viewing Reports** for more information. To export a code-based report, you use the Export method of the export you choose. For more information, see **Export a Section Report**.

XML-Based Section Report

When you add an ActiveReports 7 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)**.

Page Report Concepts

There are a number of concepts that only apply to page reports.

In this section

Page Report Toolbox

This section provides information on each of the report controls and data regions available in the **ActiveReports 7 Page Report** group in the Visual Studio toolbox.

Data Sources and Datasets

Find out 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.

Shared Data Source (RDSX)

See the advantages of using the RDSX proprietary file format as a data source in your reports.

Expressions

Learn about setting expressions in reports and creating expression through the Expression Editor.

Using Script in a Page Report

Embed code in the script tab to extend the features in your reports.

Report Dialog

See the various options provided in Report Dialog.

FixedPage Dialog

See the various options provided in FixedPage Dialog.

Grouping Data (Page Layout)

See the various options provided for grouping data in ActiveReports.

Add Page Numbering

Select out of a list of pre-defined formats or create custom formats to display page numbers in reports.

Themes

Create themes to define the appearance of reports, and apply the themes to any number of reports for a consistent look.

Rendering

Learn how to use Rendering Extensions to render page reports in various formats.

Master Reports

Use master reports to create a reusable template of common elements you can apply to other reports.

Data Visualizers

Learn about a number of ways to make your data pop using small graphs in images and background colors.

Custom Resource Locator

Find information about the ResourceLocator class that allows you to find resources on your machine for use in your CPL reports.

Page Report Toolbox

When a Page report has focus in Visual Studio, the **ActiveReports 7 Page Report** toolbox group offers a number of report controls and data regions that you can use when creating a page report. You can drag these from the toolbox and drop them onto your page 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

BandedList

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 25 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.

Calendar

The Calendar report control displays date-based data or events in a calendar format in your report. You can modify the appearance of the calendar and events.

Chart

The 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. As a container, this control serves to keep the report controls it contains together when they might otherwise grow apart due to the growth of adjacent report controls.

FormattedText

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.

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.

Matrix

The Matrix is a data region with dynamic numbers of rows and columns, and is similar in functionality to a cross tab or pivot table.

OverflowPlaceholder

The Overflow Placeholder report control is only available with FPL page reports. It is a simple rectangle that you link to a List, BandedList, Matrix, 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. It presents the most recent measurement as the right-most data point and compares it with earlier measurements to the left. 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. Please note that each subreport instance is run as a separate report, which can cause a noticeable difference in performance when processing large reports, in which case a data region may be more efficient.

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. The header and footer can be removed, and group headers and footers can be added to suit your needs. Each cell contains a TextBox by default, but you can replace the TextBox with any report control.

TextBox

The TextBox report control displays data, and is the default report control that appears in each cell of a table or matrix. It is also the report control 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.

BandedList

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.
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 CPL 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.

Tooltip: Enter the value or expression you want to appear when a user hovers the cursor over the banded list in the viewer at run time.

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 item.

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 report 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 item.

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 item: 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 39 different 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 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.

General

Name: Enter a name for the barcode that is unique within the report.

Tooltip: Enter the value or expression you want to appear when a user hovers the cursor over the barcode in the viewer at run time.

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.

BarCodeStyle	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.
Codabar	Codabar uses A B C D + - : . / \$ and numbers.
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.
Code25mat	Code 25 Matrix is a two-dimensional version of the linear Code 2 of 5 barcode.
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). If there are only 12 numbers in the string, it calculates a checksum and adds it to the thirteenth position. If there are 13, it validates the checksum and throws an error if it is incorrect.
EAN_8	EAN-8 uses only numbers (7 numbers and a check digit).
EAN128FNC1	<p>EAN-128 is an alphanumeric one-dimensional representation of Application Identifier (AI) data for marking containers in the shipping industry.</p> <p>This type of bar code contains the following sections:</p> <ul style="list-style-type: none">• Leading quiet zone (blank area)• Code 128 start character• FNC (function) 1 character which allows scanners to identify this as an EAN-128 barcode• Data (AI plus data field)• Symbol check character (Start code value plus product of each character position plus value of each character divided by 103. The checksum is the remainder value.)• Stop character• Trailing quiet zone (blank area) <p>The AI in the Data section sets the type of the data to follow (i.e. ID, dates, quantity, measurements, etc.). There is a specific data structure for each type of data. This AI is what distinguishes the EAN-128 code from Code 128.</p> <p>Multiple AIs (along with their data) can be combined into a single bar code.</p> <p>EAN128FNC1 is a UCC/EAN-128 (EAN128) type barcode that allows you to insert FNC1 character at any place and adjust the bar size, etc., which is not available in UCC/EAN-128.</p> <p>To insert FNC1 character, set “\n” for C#, or “vbLf” for VB to Text property at runtime.</p>
IntelligentMail	Intelligent Mail, formerly known as the 4-State Customer Barcode, is a 65-bar code used for domestic mail in the U.S.
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.
MSI	MSI Code uses only numbers.
Pdf417	<p>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.</p>
PostNet	PostNet uses only numbers with a check digit.
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.
RSS14Stacked	RSS14Stacked uses the EAN.UCC information with Indicator digits as in the RSS14Truncated, but stacked in two rows for a smaller width.
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 runtime.
RSSExpandedStacked	<p>RSSExpandedStacked uses the EAN.UCC information with AI elements as in the RSSExpanded, but stacked in two rows for a smaller width.</p> <p>RSSExpandedStacked allows you to insert an FNC1 character as a field separator for variable length Application Identifiers (AIs).</p>
	To insert FNC1 character, set “\n” for C#, or “vbLf” for VB to Text property at runtime.
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.
UCCEAN128	UCC/EAN –128 uses the complete ASCII character Set. This is a special version of Code 128 used in HIBC applications.
UPC_A	UPC-A uses only numbers (11 numbers and a check digit).
UPC_Eo	UPC-Eo 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_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.

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): 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 (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.

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 item. This enables the drop-down box where you can select the report 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

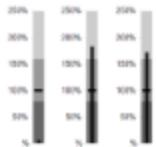
The Bullet report control is an easy-to-read linear gauge that is a good alternative to using a dashboard for data visualization.

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.

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 item 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 item. This enables the drop-down box below where you can specify the report 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.

Calendar

The **Calendar** report control is used to display date-based data or events in a calendar format in your report. In the Properties Window or the Calendar Dialog, you can modify the appearance of the days, months, weekends, and events in the calendar, and create events for it.



Calendar Dialog

Properties for the Calendar are available in the Calendar dialog. To open it, with the Calendar control selected on the report, under the Properties Window, click the **Property dialog** link.

The Calendar 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 calendar that is unique within the report.

Data

Dataset Name: Select a dataset to associate with the calendar. The list is populated with all of the datasets in the report's dataset collection.

Event Settings

Start Date: Enter an expression to use to identify the Start Date value(s) of the events to be displayed.

End Date: Enter an expression to use to identify the End Date value(s) of the events to be displayed.

Value: Enter an expression to use to identify the event text value(s) of the events to be displayed.

Detail Grouping

Name: Enter a name for the detail group that is unique within the report. A name is created automatically if you do not enter one.

Group on: Enter an expression to use for grouping the data. If you have already assigned a dataset name in the **Dataset Name** property, you can select a field from the dataset.

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.

Event Appearance

Format: 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.

Alignment: Select the horizontal alignment of the event 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.

Background

Fill Color: Select a color to use for the background of the calendar event.

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.

Image

You can display an image on all calendar events using the following options to define the image.

Source: Choose from External, Embedded, or Database.

MIME Type: Select the MIME type of the image chosen.

Value: Enter the name of the image to display.

Calendar Appearance

The Calendar Appearance page has the following tabs: Month Appearance, Day, Day Headers, Weekend, and Filler Day.

All but Day Headers have the same properties. (There is no Formatting section on the Day Headers tab.)

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: 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 Fill Color: Select a color to use for the background of the calendar's month section.

Formatting

Alignment: Select the horizontal alignment of the calendar month text.

Format: 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.

Navigation

Action

Select one of the following actions to perform when a user clicks on an event in the calendar.

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.

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 item 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 calendar. You will then be able to provide a bookmark link to this item from another report item using a **Jump to bookmark** action.

Data Output

Element Name: Enter a name to be used in the XML output for this calendar.

Output: Choose **Auto**, **Yes**, **No**, or **Content only** to decide whether to include this calendar in the XML output. **Auto** exports the contents of the calendar only when the value is not a constant.

Chart

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, 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.

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.

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.

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.

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: Enter the value or expression you want to appear when a user hovers the cursor over the chart in the viewer at run time.

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.

All Chart Types

Series label: Enter an expression to use as a series label to display in the legend.

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.

Stock Chart Type

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.

Column, Bar, Line, Pie, Area, or Doughnut Chart Types

Value: Enter an expression to use as a series value.

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, 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

The Filters tab of Category Groups page allows you to control the Filter grid collection for the group. Use 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 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

The Filters tab of Series Groups page allows you to control the Filter grid collection for the group. Use 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 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

The Filters page of the Chart Data dialog allows you to filter the data that is included in the chart. 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 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.

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.

Tooltip: Enter the value or expression you want to appear when a user hovers the cursor over the checkbox in the viewer at run time.

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 report control to display with a toggle image button. When the user clicks this report 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.

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: Enter an image to use for the background of the container.

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

When you select a checkbox next to a corner of the rectangle in the dialog, a yellow diamond appears. Drag the yellow diamond to change the shape of the corner.

Use the same radius on specified corners: Select this option to apply the same radius to all selected corners of the rectangle.

Use different radius on specified corners: Select this option to apply a different radius to each selected corner of the rectangle.

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 item. This enables the drop-down box below where you can specify the report 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.

FormattedText

The FormattedText report control can perform mail merge operations, plus it displays richly formatted text in XHTML. To format text in the FormattedText report control, enter XHTML code into the **Html** property.

Supported XHTML 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 **<body></body>** tags.

Tag	Description
<code><%MergeFieldName%></code>	Inserts a mail merge field.
<code><!-- --></code>	Defines a comment
<code><!DOCTYPE></code>	Defines the document type
<code><a></code>	Defines an anchor
<code><abbr></code>	Defines an abbreviation
<code><acronym></code>	Defines an acronym
<code><address></code>	Defines an address element
<code></code>	Defines bold text
<code><base /></code>	Defines a base URL for all the links in a page
<code><bdo></code>	Defines the direction of text display
<code><big></code>	Defines big text
<code><blockquote></code>	Defines a long quotation
<code><body></code>	Defines the body element (Required)
<code>
</code>	Inserts a single line break
<code><caption></code>	Defines a table caption
<code><center></code>	Defines centered text
<code><cite></code>	Defines a citation
<code><code></code>	Defines computer code text
<code><col></code>	Defines attributes for table columns
<code><dd></code>	Defines a definition description
<code></code>	Defines deleted text
<code><dir></code>	Defines a directory list
<code><div></code>	Defines a section in a document
<code><dfn></code>	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
	Defines an image
<ins>	Defines inserted text
<kbd>	Defines keyboard text
	Defines a list item
<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.

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.

Tooltip: Enter a string or expression that you want to appear when a user hovers the cursor over the image in the viewer at run time.

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 report 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.

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.

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.

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 report 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. A list can be nested within another list to provide multiple groupings of data.

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.

Tooltip: Enter the value or expression you want to appear when a user hovers the cursor over the list in the viewer at run time.

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.

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 report 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

The Filters page of the List dialog allows you to control the Filter collection for the list. You can also access the Filters collection by clicking the ellipsis (...) button next to the Filters property in the property grid. 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, 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.

Matrix

The Matrix data region contains columns and rows in which the data is inserted and arranged. Columns and rows in a matrix can be dynamic or static. Each cell in a matrix contains a textbox by default, but as with the other data regions, the textbox can be replaced with any other report control. Also like other data regions, the matrix also repeats each report control it contains for each row in the dataset, but unlike the others, it also repeats horizontally for dynamic columns.

Cells

When you drop a matrix data region onto your report, it is initially composed of four cells.

Corner	Column Header
Row Header	Aggregated Detail

The top left cell is the corner or label cell. You can leave this cell blank or use it to display a label for the matrix.

The top right cell is a column header and the bottom left cell is a row header. You can drag fields into these cells or use expressions to group the data.

The bottom right cell is used to aggregate the detail data. At run time, detail cells display aggregates of the intersections of columns and rows.

Data

The matrix data region is associated with a dataset. You can make this association in the **DataSetName** property in the Properties grid.

Groups

You can group the columns and rows in the matrix by opening the Property dialog, and on the **Row Groups** or **Column Groups** page, editing the **Group on** expression, or by dragging a field into the header cell.

You can nest column and row groups by right-clicking the UI above a column or to the left of row and choosing **Add Column Group** or **Add Row Group**, or simply by dragging another field onto the column or row into which you want to insert it.

To determine where new groups are displayed, when you drag a field into an existing group you can move the mouse up

or down (for columns) or left or right (for rows) to nest the new group within the existing group, or to nest the existing group within the new group.

At run time, the nested group repeats within the original group. For instance, when you drag the City field from your dataset onto a header already grouped by Country, the report displays the first country with all of its cities, then the next country with all of its cities, and so on until all of the countries in the dataset are displayed with all of their cities.

Static Rows and Columns

To insert a static row or column, right-click the detail cell and choose **Add Column** or **Add Row**. This displays a column next to or a row above or below the existing one and adds another detail cell.

Aggregates

Groups in a matrix do not subtotal data by default. To add this functionality, right-click the group header and choose **Subtotal**. This adds a cell to the right of the column, or below the row. Subtotal cells have a green triangle in the upper right corner which allows you to select the subtotal and set its properties in the property grid. .

Placement of Data

You can control where data is placed in several ways. You can change the **Direction** property of the matrix to **RTL** to cause dynamic column headers to expand left instead of right.

You can also move row headers to the right of the detail cells by changing the **GroupsBeforeRowHeaders** property of the matrix. The integer value you supply for this property equals the number of instances of the outermost column group that displays to the left of the row headers (or to the left if the **Direction** property is set to **RTL**). .

Matrix Dialog

Properties for the Matrix are available in the Matrix dialog. To open it, with the Matrix control selected on the report, under the Properties Window, click the **Property dialog** link.

The Matrix dialog lets you set properties on the report control with the following pages.

 **Note:** You can select <Expression...> within many of these properties to create an expression to determine the value.

General

Name: Enter a name for the matrix that is unique within the report. This name can be called in code.

Tooltip: Enter the value or expression you want to appear when a user hovers the cursor over the matrix in the viewer at run time.

Dataset name: Select a dataset to associate with the matrix. 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 Matrix is in its own section with regards to pagination.

Page breaks: Select any of the following options to apply to each instance of the matrix.

- Insert a page break before this matrix
- Insert a page break after this matrix
- Fit matrix on a single page if possible

Matrix column expand: Select the direction in which columns expand.

- Left to right
- Right to left

Groups before row headers: Select the number of columns to show before the row header columns begin. The default value of **0** displays the row header column to the left.

Visibility

Initial visibility

- **Visible:** The matrix is visible when the report runs.
- **Hidden:** The matrix is hidden when the report runs.
- **Expression:** Use an expression with a Boolean result to decide whether the matrix 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 list where you can select the report control that users can click to show or hide this matrix.

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 matrix. You will then be able to provide a bookmark link to this item from another report control using a **Jump to bookmark** action.

Filters

Click the plus sign button to add a filter to the matrix. Use 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, Operator, and Value.

Expression: Enter the expression to use for evaluating whether data should be included in the matrix.

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.

Row Groups

The Row Groups page of the Matrix dialog allows you to add, remove, or change the order of row groups using the plus sign, X and arrow buttons.

Click the **Add** button to add a new row group to the list 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.

Label: Enter an expression to identify an instance of the group for document map and search functions.

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. 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.

Column Groups

The Column Groups page of the Matrix dialog allows you to add, remove, or change the order of row groups using the plus sign, X and arrow buttons.

Click the **Add** button to add a new column group to the list, 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.

Label: Enter an expression to identify an instance of the group for document map and search functions.

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 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.

Data Output

Element name: Enter a name to be used in the XML output for this matrix.

Output: Choose Auto, Yes, or No to decide whether to include this matrix in the XML output. Choosing Auto exports the contents of the matrix.

Cell element name: Enter a name to be used in the XML output for the data element for the cell.

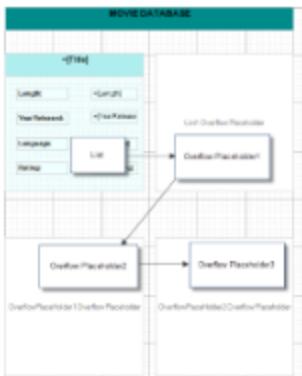
Cell element output: Choose Yes or No to decide whether to include the cell contents in the XML output.

OverflowPlaceHolder

In a fixed page layout, the OverflowPlaceHolder control is a rectangular placeholder for data that does not fit inside the fixed size of a **List**, **BandedList**, **Matrix** 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.
- **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** ('Overflow Data in a Single Page' in the on-line documentation) 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:

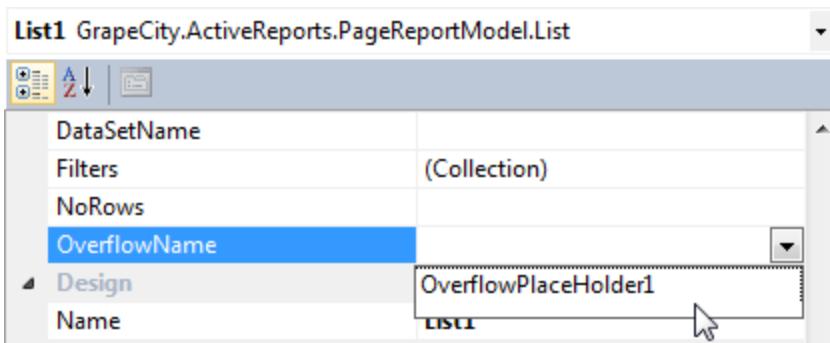
These steps assume that you have already added a Page Report template to your project, connected it to a data source and added a DataSet. See **Connect to a Data Source** and **Add a Dataset** for more information.

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 **OverflowName Property (on-line documentation)** 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.



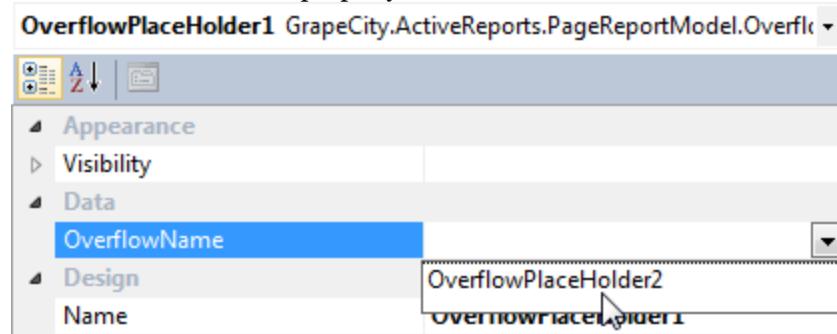
Tip: Depending on your layout requirements, you can place the OverflowPlaceHolder control on the same page tab as the data region or a different page tab.

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 the **OverflowName Property (on-line documentation)** 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.

If you choose **RoundRect**, you can use the **RoundingRadius** property to select how much to round the corners.

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.

Shape Style: Choose **Rectangle**, **RoundRect** or **Ellipse**. If you choose **RoundRect**, a preview of the shape appears below where you can select whether to use the same radius or a different radius on the corners that you select with the adjacent checkboxes. Drag the yellow diamonds to set the amount of rounding to use.

Appearance

Background

Color: Select a color to use for the background of the Shape.

Image: Enter an image to use for the background of the Shape.

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 report 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.

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 report item 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

The Filters page of the Sparkline dialog allows you to control the Filter grid collection for the sparkline. 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, 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, we recommend that you use data regions instead of Subreport controls wherever possible for better performance. 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 **Work with Report Controls and Data Regions**.

 **Note:** The Subreport control is not supported in FPL Page reports, only CPL. Also, you cannot use a Section report as the target of a Subreport in a Page report, and vice versa. But you can export any type of report to RDF and use that as the target of a Subreport in a section report.

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.

Tooltip: Enter the value or expression you want to appear when a user hovers the cursor over the subreport in the viewer at run time.

Subreport: Select the report to display within the subreport control. The combo box is populated with all of the reports in the local folder. Alternatively, you can drag a report from the Solution Explorer and drop it onto the report surface to create a new subreport.

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 report 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.

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 CPL reports, you can embed other data regions in table cells.

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 CPL 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. This is useful when you need to create, for example, a report which is grouped by country. You just add a group, and in the **Group On** expression, choose the Country field from your dataset.

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 **Grouping in a Data Region**.

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, background color. You can use graphical elements within the cells and borders on the cells themselves.

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.

Tooltip: Enter the value or expression you want to appear when a user hovers the cursor over the table in the viewer at run time.

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.

Has own page numbering: Select to indicate whether this BandedList is in its own section with regards to pagination.

Page breaks: Select any of the following options to apply to each instance of the table.

- Insert a page break before this table
- Insert a page break after this table
- Fit table on a single page if possible

Header and Footer: Select any of the following options.

- Repeat header row on each page
- Repeat footer row on each 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 item: 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.

Keep together on one page if possible: Keep together the repeated sections on one 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.

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 item: 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.

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

The Filters page of the Table dialog allows you to control the Filter grid collection for the table. Use 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, 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.

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 Matrix 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.

The **HideDuplicates** property allows you to specify a grouping or dataset within which to hide the TextBox when its value equals the value of the same item in the previous row.

By default, in CPL 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 FPL 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.
Matrix data region, corner cell	=Sum(Fields!SalesAmount.Value)	Displays a summary of the sales amount for the dataset associated with the Matrix.
Matrix data region, column header cell	=Fields!SalesAmount.Value	Displays the value at the top of a new column for each row of data running to the right.
Matrix data region, row header cell	=Fields!SalesAmount.Value	Displays the value to the left of a new row for each row of data running down the page.
Matrix data region, detail 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.

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.

Tooltip: Enter the value or expression you want to appear when a user hovers the cursor over the textbox in the viewer at run time.

Note: You must select **Print Preview** mode in order to see this property working in the viewer.

Value: Enter an expression or a static label, or choose a field expression from the drop-down list.

Hide duplicates: This option is disabled unless the TextBox is within a data region.

Data region: Enter or select a data grouping in which to hide repeated values.

 **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 report 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.

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.

Line Height: This property sets the height of lines of text with leading and ascending space.

 **Note:** This property only affects HTML output.

Character Spacing: This property sets the space between characters.

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>

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

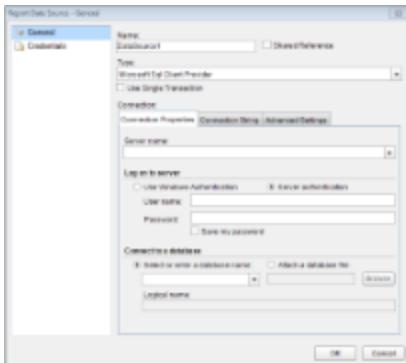
A dataset fetches data from the data source to display in a report. The **DataSet Dialog** is where you provide a command type and query string and choose other options for your dataset.

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.

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.
- 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
 - **DataSet Provider**
 - Microsoft ODBC Provider
 - Microsoft OleDb Provider
 - **Object Provider**
 - Oracle Client 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, OleDb, or Oracle as the data source **Type** value, the **Connection Properties**, **Connection String** and **Advanced Settings** pages appear under the **Connection** section. In other data source types, only the **Connection String** page appears.

Connection Properties

The Connection Properties page gives access to properties specific to the following data types.

SQL

- **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
- **Connect to a database:** Through this field, you can select whether to enter a database name or attach a database file.
- Below this field you can also check the **Save My Password** option for future reference.

OleDb

- **OLE DB Provider:** This field requires you to select one among the list of OLE DB Providers provided in the drop down list.
- **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.

Oracle

- **Server name:** This field requires you to enter a server name.
- **Log on to server:** This field requires you to enter a user name and a password for server authentication.
- Below this field you can also check the **Save My Password** option for future reference.

Connection String

Enter a connection string in the provided text box, or use the Connection Properties page to create one for SQL, OleDB, or Oracle data types.

Following is the sample connection string to access the Northwind database in the C directory:

Sample connection string

```
provider=Microsoft.Jet.OLEDB.4.0;data source=c:\nwind.mdb;
```

Advanced Settings

The Advanced Settings page gives access to properties specific to each data type.

SQL

With the SQL data type, the Advanced Settings page 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.

OleDB

With the OleDB data type, the Advanced Settings page 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_JETOLEDDB_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.

 **Note:** You can see the description of each property at the bottom of the dialog when the property is selected. To provide more room for longer descriptions, drag the grab handle above the description.

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.

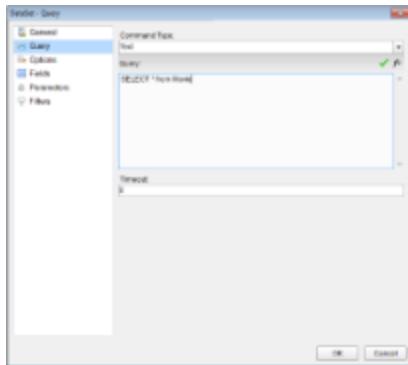
No credentials

Select this option only if the data in the report is for general public consumption.

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.

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 and Oracle data providers. To see 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.

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, Oracle, OleDb	The field value is the name of a field returned by the 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.	Quantity OrdersToOrderDetails.CustomerID
XML	The field value is an XPath expression that returns a value when evaluated with the query.	./OrderQuantity ../OrderInfo/OrderDate
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	Quantity Order.Customer.FirstName

object returned from a property.

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 comprises 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.

Shared Data Source (RDSX)

In ActiveReports, a shared data source refers to a file in RDSX format that contains data connection information. RDSX (Report Data Source XML) is a proprietary file format that functions as a data source for a single report or multiple

reports.

You can use this data source in page reports. See **Create and Edit a Shared Data Source** for information on how to create a new shared data source or modify an existing one.

Advantages of a Shared Data Source

- It is a reusable data connection that you can use in a single report or multiple reports.
- It is a separate file in RDSX format that you can access from any page report, move to different folders, and rename.
- It sets the data connection in a report so you need only add a SQL query to create a data set.
- It lets you update the connection string in one place for all reports referencing the Shared Data Source.
- You can use it with any data connection type.

 **Caution:** An RDSX file contains the connection string, but you cannot define a data set in it. See **Add a Dataset** for information on creating a data set.

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,

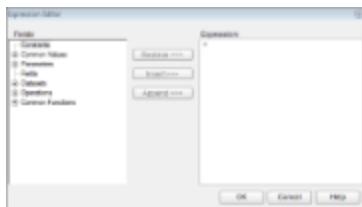
- Properties in the properties window
- Expression Editor Dialog

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 seven 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.

To create an Expression in the Expression Editor

The Expression Editor dialog comprises 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.

Common Functions

In page reports, 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.

- DateAdd
- DateDiff
- DatePart
- DateSerial
- DateString
- DateValue
- Now
- Today
- Day
- Hour
- Minute
- Month
- MonthName
- Second
- TimeSerial
- TimeValue
- TimeOfDay
- Timer
- TimeSpan
- Weekday
- WeekdayName
- Year

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.

- Abs
- Acos
- Asin
- Atan
- Atan2
- BigMul
- Ceiling
- Cos
- Cosh
- E
- Exp
- Fix
- Floor
- IEEERemainder
- Log
- Log10
- Max
- Min
- PI
- Pow
- Round
- Sign
- Sin
- Sinh
- Sqrt
- Tan
- Tanh

Inspection

These are all methods from the Information class in Visual Basic. Please see the msdn [Information Class](#) topic for more information.

- IsArray
- IsDate
- IsDBNull
- IsError
- IsNothing
- IsNumeric

Program Flow

These are all methods from the Interaction class in Visual Basic. Please see the msdn [Interaction Class](#) topic for more information.

- Choose
- IIf
- Switch
- Partition

Aggregate

You can use aggregate functions within report control value expressions to accrue data. ActiveReports Developer 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	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(Fields!Discontinued.Value=True, Sum, Fields!InStock.Value)
Avg	Calculates the average of the non-null values returned by the expression.	=Avg(Fields!Cost.Value, Nothing)
Count	Calculates the number of non-null values returned by the expression.	=Count(Fields!EmployeeID.Value, Nothing)
CountDistinct	Calculates the number of non-repeated values returned by the expression.	=CountDistinct(Fields!ManagerID.Value, "Department")
CountRows	Calculates the number of rows in the scope returned by the expression.	=CountRows("Department")
DistinctSum	Calculates the sum of the values returned by an expression using only the rows when the value of another expression is not	=DistinctSum(Fields!OrderID.Value, Fields!OrderFreight.Value, "Order")

Function	Description	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(Fields!Discontinued.Value=True, Sum, Fields!InStock.Value)
Avg	Calculates the average of the non-null values returned by the expression.	=Avg(Fields!Cost.Value, Nothing)
Count	Calculates the number of non-null values returned by the expression.	=Count(Fields!EmployeeID.Value, Nothing)
CountDistinct	Calculates the number of non-repeated values returned by the expression.	=CountDistinct(Fields!ManagerID.Value, "Department")
CountRows	Calculates the number of rows in the scope returned by the expression.	=CountRows("Department")
DistinctSum	Calculates the sum of the values returned by an expression using only the rows when the value of another expression is not	=DistinctSum(Fields!OrderID.Value, Fields!OrderFreight.Value, "Order")

	repeated.	
First	Shows the first value returned by the expression.	<code>=First(Fields!ProductNumber.Value, "Category")</code>
Last	Shows the last value returned by the expression.	<code>=Last(Fields!ProductNumber.Value, "Category")</code>
Max	Shows the largest non-null value returned by the expression.	<code>=Max(Fields!OrderTotal.Value, "Year")</code>
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.	<code>=Median(Fields!OrderTotal.Value)</code>
Min	Shows the smallest non-null value returned by the expression.	<code>=Min(Fields!OrderTotal.Value)</code>
Mode	Shows the value that appears most frequently in the values returned by the expression.	<code>=Mode(Fields!OrderTotal.Value)</code>
RunningValue	Shows a running aggregate of values returned by the expression (Takes one of the other aggregate functions as a parameter),	<code>=RunningValue(Fields!Cost.Value, Sum, Nothing)</code>
StDev	Calculates the dispersion (standard deviation) of all non-null values returned by the expression.	<code>=StDev(Fields!LineTotal.Value, "Order")</code>
StDevP	Calculates the population dispersion (population standard	<code>=StDevP(Fields!LineTotal.Value, "Order")</code>

	deviation) of all non-null values returned by the expression.	
Sum	Calculates the sum of the values returned by the expression.	=Sum(Fields!LineTotal.Value, "Order")
Var	Calculates the variance (standard deviation squared) of all non-null values returned by the expression.	=Var(Fields!LineTotal.Value, "Order")
VarP	Calculates the population variance (population standard deviation squared) of all non-null values returned by the expression.	=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.

- ToBoolean
- ToByte
- ToDateTime
- ToDouble
- ToInt16
- ToInt32
- ToInt64
- ToSingle
- ToUInt16
- ToUInt32
- ToUInt64

Miscellaneous

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. Please see the msdn [Built-in Functions for ReportViewer Reports](#) topic for more information. GetFields is a proprietary function to extend RDL specifications.

Function Description

InScope	Determines whether the current value is in the indicated scope.
Level	Returns the level of the current value in a recursive hierarchy.
Previous	Returns the previous value within the indicated scope.
RowNumber	Shows a running count of all the rows in the scope returned by the expression.
GetFields	Returns an IDictionary<string,Field> object that contains the current contents of the Fields collection. Only valid when used

Syntax and Example

=InScope(<Scope>)
=InScope("Order")
=Level(<i>optional <Scope></i>)
=Level()
=Previous(<Expression>)
=Previous(Fields!OrderID.Value)
=RowNumber(<i>optional <Scope></i>)
=RowNumber()
=GetFields()
=Code.DisplayAccountID(GetFields())

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.

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
```

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.

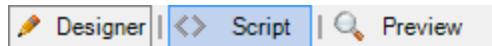
 **Note:** You cannot set the Scope parameter to Nothing outside of a data region.

Using Script in a Page Report

In a page 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 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, 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.

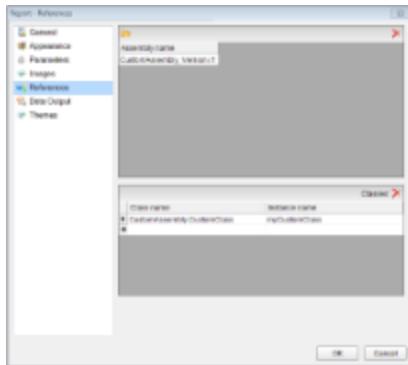
Create custom assemblies

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.
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 ...\\Common Files\\ComponentOne\\ActiveReports Developer 7 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.

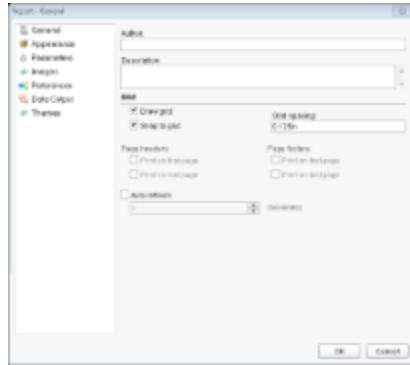
- To access static members (member denoted as `public static` in C# assemblies, or as `Public Shared` in Visual Basic assemblies):
`=Namespace.Class.Member`
- To access class instances:
`=Code.InstanceName`

Report Dialog

In a page 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 Continuous Page Layout (CPL) 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 Continuous Page Layout (CPL) 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.

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.

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.

 **Caution:** If you are using the ActiveReports Standalone Report designer application, make sure the assembly you

load is created using .NET 3.5 framework or below as the Standalone Report designer is a .NET 3.5 application.

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.

 **Note:** Any assemblies you reference must be copied to the ActiveReports Developer 7 folder (...Common Files\ComponentOne\ActiveReports Developer 7) on your development machine as well as on any server to which you deploy your reports.

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>

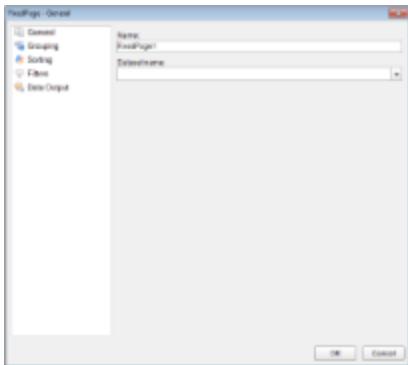
Themes

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.

FixedPage Dialog

In a Fixed Page Layout (FPL), 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 fixed page layout. 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 fixed page layout. 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 **Grouping in a FixedPage** 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.

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 Numbering](#) for further information on setting page numbering in Fixed Page Layout.

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 fixed page layout.
- **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.

Grouping Data (Page Layout)

In a page 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 **BandedList**, 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 and even nest Lists for more levels of grouping.
- In a **Matrix**, you can add columns and rows either dynamically or manually to group data.
- In a **Chart**, you can group data by categories or series.

See **Grouping in a Data Region** for further information.

 **Note:** In this topic, the data displayed in the screenshots is from the Movie table available in the Reels database. By default, the Reels.mdb file is located in the [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data folder.

In a fixed page layout, you can also group data on a report page. See **Grouping in a FixedPage** for further information.

MOVIE RELEASES		
Title	Year Released	User Rating
Wise as Fools	1963	5.1
The Last Word: Journey's End	1963	5.8
Laurel and Hardy	1963	5.8
Air Force One	1997	5.2
An Officer in the Family	1968	6.6
Good Will Hunting	1997	6.7
My Best Friend's Wedding	1997	5
Titanic	1997	9.2
Braveheart	1995	5.9
Battalion & Rollin'	1995	5.4
Gangs of New York	1995	7.4
Sweeney	1995	7
Cos' Air	1997	6.8
Contract	1997	7.8

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.

RECURSIVE HIERARCHY				
Title	Salary	ManagementRole	DepartmentID	PriorityLevel
President	30000	Senior Management	1	0
VIP Customer Manager	17000	Senior Management	1	1
VIP Customer Systems	18000	Senior Management	2	1
HQ Information Systems	18000	Middle Management	2	0
VIP Human Resources	18000	Senior Management	3	1
HQ Human Resources	18000	Middle Management	4	0
VIP Finance	17000	Senior Management	5	1
HQ Finance and Accounting	18000	Middle Management	5	0
Store Manager	18000	Store Management	6	2
Store Assistant Manager	18000	Store Management	6	3
Store Shift Supervisor	18000	Store Management	6	4
Store Cashier	18000	Store/Full Time Staff	8	0
Store Stocker	18000	Store/Full Time Staff	9	3
Store Information Systems	18000	Store/Full Time Staff	7	4
HQ Marketing	18000	Middle Management	9	1

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 + (Level0*10)) & "pt"
```

RECURSIVE HIERARCHY WITH LEVEL FUNCTION			
Node	Node ID	Management Level	Depended By
President	30401	Senior Management	1
VP Country Manager	110001	Senior Management	1
VP Information Systems	80001	Senior Management	2
HQ Information Systems	80002	Middle Management	2
VP Human Resources	90001	Senior Management	4
HQ Human Resources	90002	Middle Management	4
VP Finance	170001	Senior Management	5
HQ Finance and Accounting	38001	Middle Management	5
Store Manager	60001	Store Management	6
Store Assistant Manager	18001	Store Management	6
Store Shift Supervisor	48001	Store Management	6
Store Cashier	48002	Store Full Time Staff	6
Store Receiver	28001	Store Full Time Staff	9
Store Information Systems	38002	Store Full Time Staff	1
HQ Marketing	38001	Middle Management	3

Add Page Numbering

You can choose the page numbering format for your Page reports by selecting from a list of pre-defined formats or by creating a custom page numbering expression.

Adding page numbers to a Page report

There are two ways to add page numbering to a page 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 Description

Format

- | | |
|-----------------------------|---|
| 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: <code>="Page " & Globals!PageNumber & " of " & Globals!TotalPages</code> |
| 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: <code>="Page " & Globals!PageNumberInSection & " of " & Globals!TotalPagesInSection</code> |
| 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: <code>="Page " & Globals!CumulativePageNumber & " of " & Globals!CumulativeTotalPages</code> |
| Page Number | This format displays only the current page number of a report. Use the following expression to set this page numbering format: <code>=Globals!PageNumber</code> |
| Page Number | This format displays only the current page number of a specific grouped report section. Use the following |

(Section)	expression to set this page numbering format: <code>=Globals!PageNumberInSection</code>
Total Pages	This format displays only the total number of pages in the report. Use the following expression to set this page numbering format: <code>=Globals!TotalPages</code>
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: <code>=Globals!TotalPagesInSection</code>
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: <code>=Globals!CumulativePageNumber</code>
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: <code>=Globals!CumulativeTotalPages</code>

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."

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 **CollateBy Enumeration (on-line documentation)** 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.

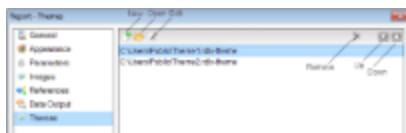
In the **Theme Editor**, you can create a new theme by setting colors, fonts, images, and **Use Constant Expressions in a Theme** and then saving a new theme as an .rdlx-theme file on your local machine. Then you can add this theme to your report in the **Report - Themes** dialog. Also, in the **File** 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 > ComponentOne > ActiveReports Developer** and select **ActiveReports Developer 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.

Rendering

In a Page Layout, you can use the **Render** method in **Rendering Extensions** of the **PageDocument** class to render in any one of the following formats.

HTML

HTML, or hypertext markup language, is a format that opens in a Web browser. It is a good format for delivering content because virtually all users have an HTML browser. You can use the **HTMLRenderingExtension** (**'HtmlRenderingExtension Class' in the on-line documentation**) to render your report in this format.

In order to render your report in HTML, add reference to the following assemblies in the project:

- GrapeCity.ActiveReports.Export.Html.v7.dll
- GrapeCity.ActiveReports.Extensibility.v7.dll

Note: Rendering to HTML requires the .NET Framework full profile version. To ensure you are using the full profile version, go to the Visual Studio **Project** menu > **Properties** > **Compile** tab > **Advanced Compile Options** (for Visual Basic projects) or to the **Project** menu > **Properties** > **Application** tab (for C# projects) and under **Target framework** select a full profile version.

Limitations

- HTML is not the best format for printing. Use the PDF rendering extension instead.
- Vertical-align style is always Top for text boxes in HTML reports. If Bottom or Middle vertical alignment is an essential style element in your report, it is recommended to use any format other than HTML.

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.

PDF

Portable Document Format (PDF), is a format recommended for printing and exports. 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 such features as font linking, digital signatures and end-user defined characters.

In order to render your report in PDF, add reference to the following assemblies in the project:

- GrapeCity.ActiveReports.Export.Html.v7.dll
- GrapeCity.ActiveReports.Extensibility.v7.dll

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.

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.

Image

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, EMF, GIF, JPEG, TIFF, and PNG.

In order to render your report in an Image format, add reference to the following assemblies in the project:

- GrapeCity.ActiveReports.Export.Image.v7.dll
- GrapeCity.ActiveReports.Extensibility.v7.dll

Interactivity

As the output is an image file, reports rendered in this format do not support interactivity. Any hidden data at the time of rendering is rendered as hidden in the image. Therefore, in order to display all hidden data (like in a drill-down report), expand all toggle items prior to rendering.

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.

In order to render your report in XML, add reference to the following assemblies in the project:

- GrapeCity.ActiveReports.Export.Xml.v7.dll
- GrapeCity.ActiveReports.Extensibility.v7.dll

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.

Word

MS Word is one of the formats to which you can render your report using the **WordRenderingExtension** (**'WordRenderingExtension Class' in the on-line documentation**).

In order to render your report in MS Word, add reference to the following assemblies in the project:

- GrapeCity.ActiveReports.Export.Word.v7.dll
- GrapeCity.ActiveReports.Extensibility.v7.dll

Limitations

Although background colors for controls export to Word documents, background colors for sections such as Body and Page Header or Footer do not.

Interactivity

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. And If a Document map is essential to your report, it is recommended to use the PDF rendering extension.

Master Reports

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.

Advantages of a Master Report

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

Designing Master Reports

When designing a master report, you use controls, code, data sources, and layout properties in the same way like 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 a page report, you can create a master report in a Continuous Page Layout (CPL) where the master report is converted to the RDLX-master file format that you can apply like a template to content reports.

A ContentPlaceHolder control appears in the toolbox when you convert a CPL report to a Master Report. This control provides the region to use for creating or designing a content report after applying a master report template.

 **Note:** In a section report (code-based report), you have 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 on 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. In the case of 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 as any other report definition.

Data Visualizers

The Image and TextBox report controls support a type of expression called Data Visualizers that allow you to create small graphs to make your data easier to understand. For example, you can red flag an overdue account using the Flags Icon Set as a background image. There are several types of Data Visualizers available through a dialog linked to properties on the Image and TextBox report controls.

Image Data Visualizers

These Data Visualizers are supported in the Image report control **Value** property, and also in the TextBox report control **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 about using 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.

Background Color Data Visualizer

These Data Visualizers are available in the TextBox report control **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 TextBox BackgroundImage, or as an Image Value. You can use the standard image strips included with ActiveReports, or create custom image strips.

Standard Image Strips

Name	Image
Checkbox	
3TrafficLights	
Arrows	
Blank	
Flags	
GrayArrows	
Quarters	
Rating	
RedToBlack	
Signs	
Symbols1	
Symbols2	
TrafficLights	

To use these image strips, place code like the following in the **BackgroundImage** property of a TextBox report control, or in the **Value** property of an Image report control.

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)
```

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	红旗
1001	5	5	0	红旗
1002	14	6	8	黄旗
1003	15	4	11	绿旗
1004	3	10	-7	红旗
1005	5	8	-3	红旗
1006	0	2	-2	红旗
1007	6	7	-1	绿旗
1008	7	5	2	黄旗
1009	8	1	7	黄旗
1010	1	1	0	红旗
1011	2	9	-7	红旗
1012	4	1	3	黄旗
1013	0	8	-8	红旗
1014	17	7	10	黄旗
1015	19	5	14	绿旗
1016	11	5	6	黄旗

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

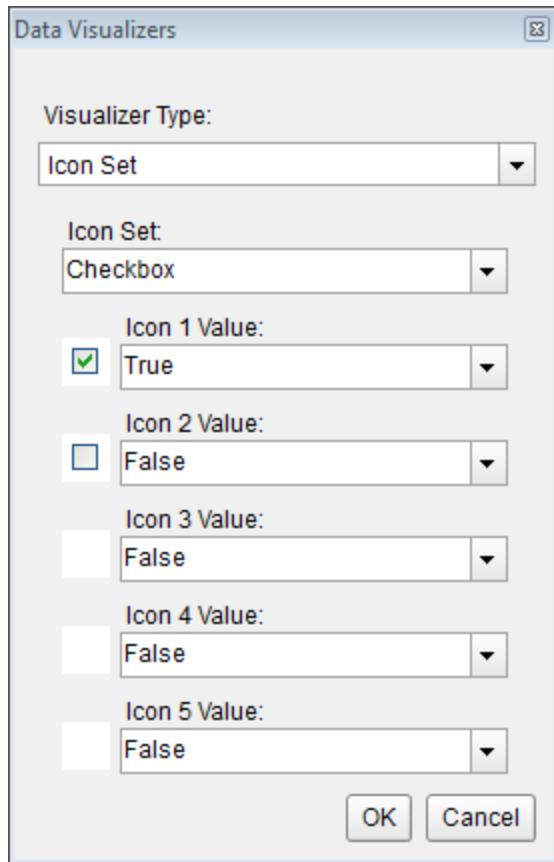
```
=IconSet("theme:ThemeImageName", condition#1, condition#2, condition#3, condition#4, condition#5)
```

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 a TextBox report control, or the **Value** property of an 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).

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 an expression with this syntax in the **BackgroundImage Value** property of a TextBox or the **Value** property of an Image. This renders 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:

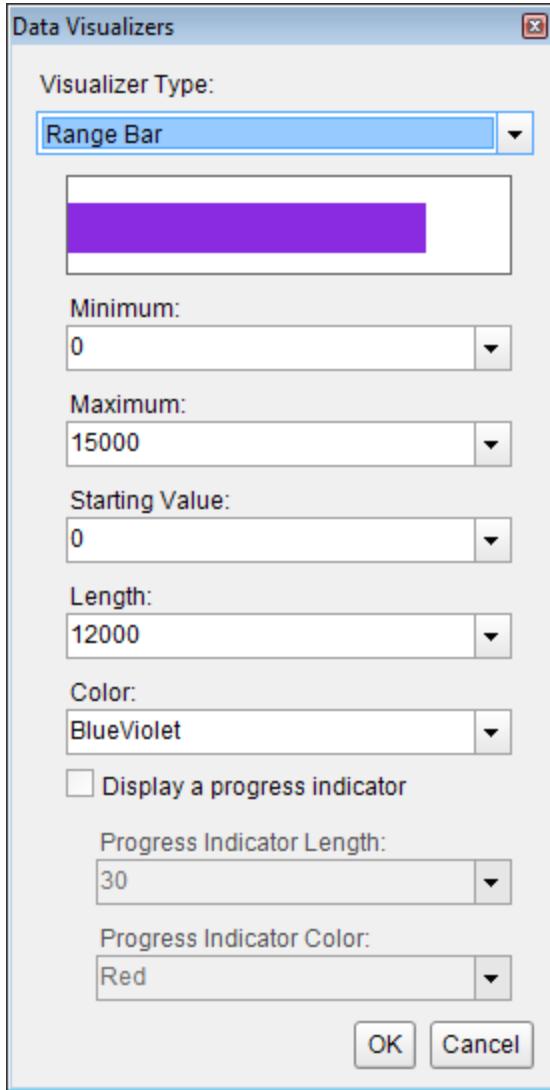
1. If the **Start** value plus the **Length** value is less than the **Minimum** value, **Length** becomes **Minimum** minus **Start**.
2. If the **Start** value plus the **Length** value is greater than the **Maximum** value, **Length** becomes **Maximum** minus **Start**.

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 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).

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 an expression with this syntax in the **BackgroundImage** property of a TextBox or the **Value** property of an Image. This renders 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.

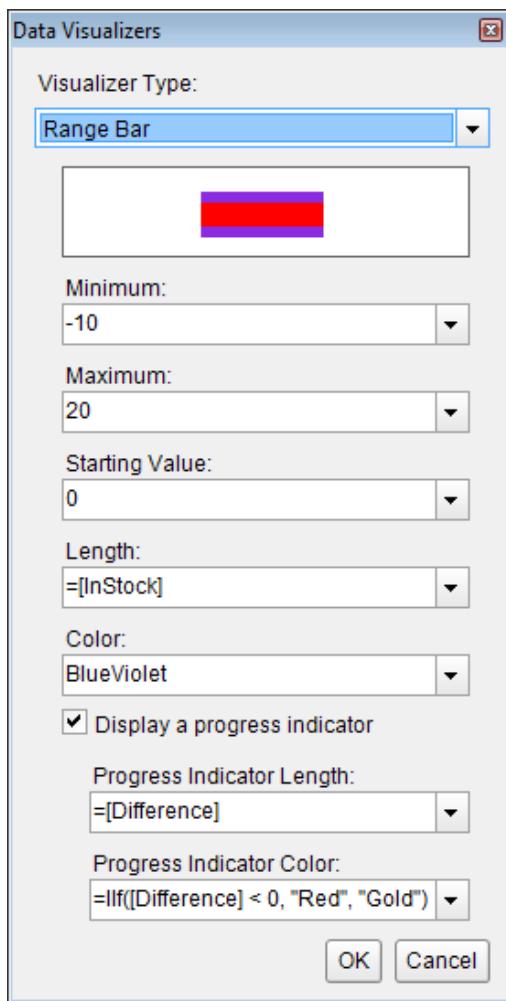
Parameter	Default Value
Minimum	0
Maximum	0
Color	Green
Start	0
Length	0
ProgressColor	Red
Progress	0

Dialog

When you select a TextBox 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).

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

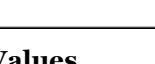
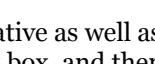
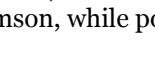
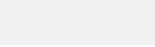
Use an expression with this syntax in either the **BackgroundImage Value** property of a TextBox or the **Value** property of an Image. This renders a bar in the color specified, the length of which changes depending on the number returned by the Value parameter, in the case of the simple example, InStock. 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:

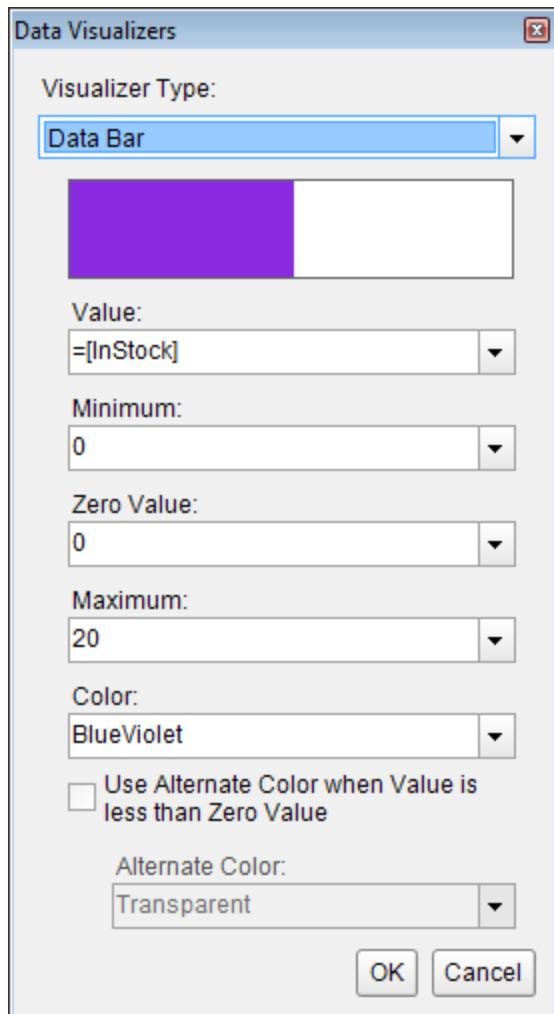
1. The **Maximum** is less than or equal to the **Minimum**.
2. The expression is placed in a property which does not take an image.
3. The **Source** property of the image is not set to **Database**.

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	<i>null</i>

Dialog

When you select a TextBox 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.



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.



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 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**.

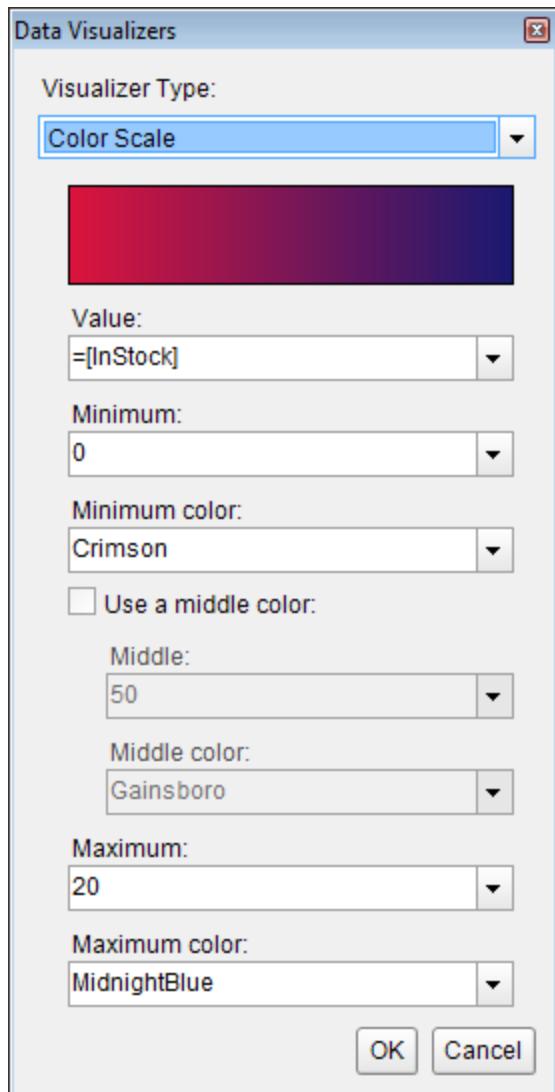
If the argument for any of the parameters cannot be converted to the required data type, the default value is used instead.

Parameter	Default Value
Value	0
Minimum	0
Maximum	0
StartColor	Silver
EndColor	WhiteSmoke

Dialog

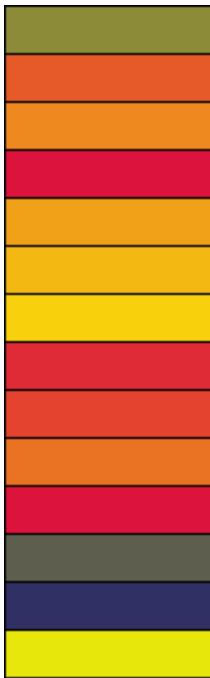
When you select a TextBox control on your report, in the Properties window or Properties dialog, you can drop down the **BackGroundColor** property and select <**Data Visualizer...**> to launch the dialog. To build the data visualizer expression, select the appropriate values for each of the options 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 TextBox background color in a range of colors to indicate minimum, middle, and maximum values, and all shades in between.



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 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**.

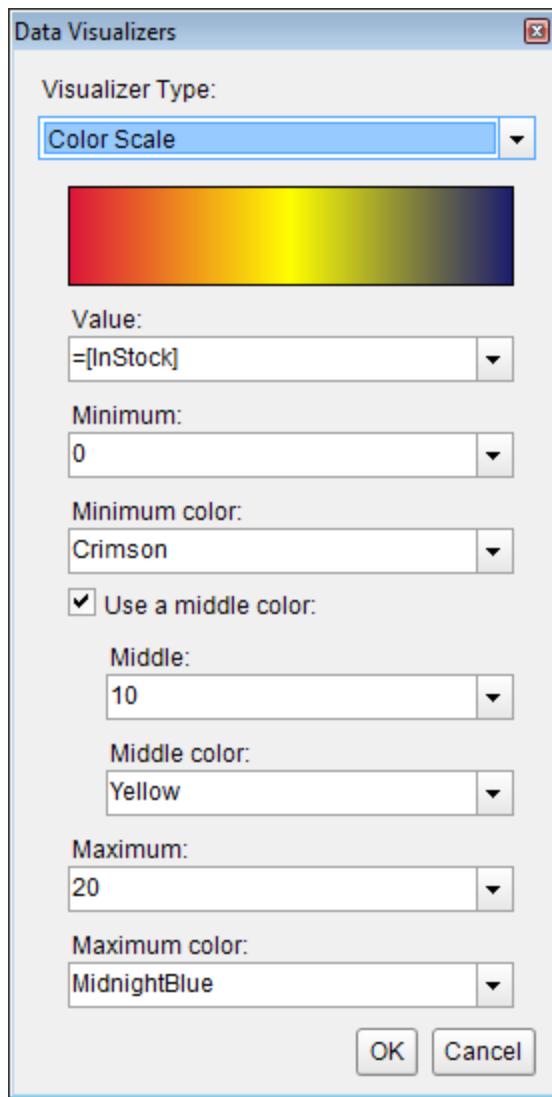
If the argument for any of the parameters cannot be converted to the required data type, the default value is used instead.

Parameter	Default Value
Value	0
Minimum	0
Middle	0
Maximum	0
StartColor	Silver
MiddleColor	Gainsboro
EndColor	WhiteSmoke

Dialog

When you select a TextBox 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 can resolve resources from your file system using file paths, but sometimes resources are preserved in very specific sources, such as a database. With CPL page reports, 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 **ResourceLocator Class (on-line documentation)** 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.Extensibility;
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:";
        /// Obtains and returns the resource, or returns null if it is not found.
        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;
        }
        /// Returns a stream containing the specified image from the My Pictures
        folder, or null if the picture is not found.
        /// The path parameter is the URI of the image located in My Pictures,
        e.g. MyImages:logo.gif
        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 GrapeCity.ActiveReports.Extensibility
Imports System.Globalization
Imports System.IO
Imports System.Runtime.InteropServices

Public Class MyPicturesLocator
    Inherits ResourceLocator

    Const UriSchemeMyImages As String = "MyPictures:"

    ' Obtains and returns the resource, or null if it is not found.
    ' The resourceId parameter contains the information about the resource to
obtain.

    Public Overrides Function GetResource(ByVal resourceId As ResourceInfo) As
Resource
        Dim resource As Resource
        Dim name As String = resourceId.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
End Function
```

```
    Return stream  
End Function
```

```
End Class
```

Sample

In the Samples folder installed with the product, there is a Custom Resource Locator sample in a path like:

```
C:\Users\MyUserName\Documents\ComponentOne Samples\ActiveReports Developer 7\Page  
Reports\CPL\API\C#\CustomResourceLocator
```

This sample 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 7 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.

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.

Grouping Data in Section Reports

Learn about grouping data in section layout.

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 7 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 **Page 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

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 (Section 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 (Section Report)

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.

PageBreak

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 over 30 barcode styles and bind it to data.

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.

OleObject

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.

ChartControl

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.

ReportInfo

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.
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.

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.

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.
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.

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.

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.

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**.

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.

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)

To load a file into the report at run time, use the Load method. For more information, see **Load Method (on-line documentation)**.

Supported Tags

HTML Tags

The following HTML tags are supported in the RichTextBox control.

 **Tip:** In order to show special characters in an HTML file loaded into the control, use the character entity reference (for example, è for è or & for &).

Tag	Description	Attributes
	Bold	none
<I>	Italic	none
<P>	Paragraph	
	Strong (looks like bold)	none
<BIG>	Big	none
<SMALL>	Small	none
<PRE>	Preformatted	none
	Font	face, size, color, style (see notes for style attributes)
<BODY>	The body tag	background, text, leftmargin
<H1> - <H6>	Heading levels one through six	none
 	Line break	none
	Emphasized (looks like Italics)	none
<U>	Underlined	none
	Image	
<SUP>	Superscript	none
<SUB>	Subscript	none
<CENTER>	Center alignment	none
<TABLE>	Table	
<TR>	Table row	
<TH>	Table head	none
<TD>	Table datum	
	List item	none (nested levels automatically use disc, then circle, then square bullets)
	Ordered list	type
	Unordered list	type
<STRIKE>	Strike through	none

Style Attribute Properties

The style attribute of , <P>, and <TABLE> tags supports the following properties.

- border-bottom
- border-bottom-width
- border-color
- border-left
- border-left-width
- border-right
- border-top-width
- border-width
- font-family
- font-size
- height
- line-height
- margin-top
- padding-bottom
- padding-left
- padding-right
- padding-top
- table-layout

- border-right-width
- border-style
- border-top
- margin-bottom
- margin-left
- margin-right
- text-align
- text-indent
- width

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 and in XML exports.

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. In the **Style** property, you can select whether to display it as an ellipse, a rectangle, or a rounded rectangle.

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 in pixels.
Style	Gets or sets the shape type to draw.
RoundingRadius	Sets the radius of each corner for the RoundRect shape type.

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.

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%.

Rounding radius: To enable this property, set the Shape type to RoundRect. Set the percentage of rounding you want to apply to the radius of each corner of the shape, where 0% is perfectly square and 100% is perfectly round.

Line style: Select a line style to use for the shape line. You can set it to Transparent, Solid, Dash, Dot, DashDot, or DashDotDot.

Line weight: Enter the width in pixels 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.

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.
LineWidth	Gets or sets the pen width of the border line in pixels.
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.

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, or DashDotDot.

Line weight: Enter the width in pixels 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.

LineWidth Gets or sets the pen width of the line in pixels.

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.

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, or DashDotDot.

Line weight: Enter the width in pixels 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.

PageBreak

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.

 **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.

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 39 different 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, 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.

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.
BarWidth	Set the width, in inches, of the barcode's narrow bars. Setting the width to 0 expands the barcode to fit the control. The width ratio is 1 to 0.012 inches. So setting the BarWidth to 2 will have a value of 0.024 inches, while a value of 10 yields a bar width of 0.12 inches for the narrowest bars.
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.
Code128	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. <ul style="list-style-type: none">• 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.
Code49	Code49 options include Grouping and Group. If Grouping is set to True, any value not

expressed by a single barcode is expressed by splitting it into several barcodes, and the Group property may be set to a number between 0 and 8. The default values are False and 0, respectively. 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.

DataMatrix	DataMatrix options include EccMode, Ecc200SymbolSize, Ecc200EncodingMode, Ecc000_140SymbolSize, StructuredAppend, StructureNumber, and FileIdentifier. Select from supplied values, or enter a number for the StructureNumber and FileIdentifier.
Direction	Specify the print direction of the barcode symbol. Select from LeftToRight (the default value), RightToLeft, TopToBottom, or BottomToTop.
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 in pixel for SectionReports and in point/inch/cm for PageReports.
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.
PDF417	PDF417 Options only apply to the barcode style PDF417. <ul style="list-style-type: none">Column sets column numbers for the barcode. Values for this property range from 1 to 30. The default value is -1 which automatically determines row numbers.ErrorLevel sets 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.Row sets row numbers for the barcode. Values range between 3 and 90. The default value is -1 which automatically determine row numbers.Type sets the barcode type to Normal or Simple. Simple is the compact type in which the right indicator is neither displayed nor printed.
QRCode	QRCode Options only apply to the barcode style QRCode. <ul style="list-style-type: none">Connection allows any value which cannot be expressed by a single barcode to split into several barcodes. This property is used in conjunction with the ConnectionNumber property.ConnectionNumber 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.ErrorLevel values are L (7% restorable), M (15% restorable), Q (25% restorable), and H (30% restorable). The higher the percentage, the larger the barcode becomes.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.<ul style="list-style-type: none">Mask000 (i+j) mod 2 = 0Mask001 i mod 2 = 0Mask010 j mod 3 = 0Mask011 (i+j) mod 3 = 0Mask100 ((i div 2)+(j div 3)) mod 2 = 0Mask101 (ij) mod 2 + (ij) mod 3 = 0Mask110 ((ij) mod 2 +(ij) mod 3) mod 2 = 0

- Mask111 $((ij)\bmod 3 + (i+j) \bmod 2) \bmod 2 = 0$
- Model sets Model1, the original model, or Model2, the extended model.
- 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.

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.
RssExpandedStacked	Sets the number of stacked rows to use.
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.

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.

BarCodeStyle	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.
Codabar	Codabar uses A B C D + - : . / \$ and numbers.
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.
Code25mat	Code 25 Matrix is a two-dimensional version of the linear Code 2 of 5 barcode.
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). If there are only 12 numbers in the string, it calculates a checksum and adds it to the thirteenth position. If there are 13, it validates the checksum and throws an error if it is incorrect.
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 type of bar code contains the following sections: <ul style="list-style-type: none">• Leading quiet zone (blank area)• Code 128 start character• FNC (function) 1 character which allows scanners to identify this as an EAN-128 barcode

- Data (AI plus data field)
- Symbol check character (Start code value plus product of each character position plus value of each character divided by 103. The checksum is the remainder value.)
- Stop character
- Trailing quiet zone (blank area)

The AI in the Data section sets the type of the data to follow (i.e. ID, dates, quantity, measurements, etc.). There is a specific data structure for each type of data. This AI is what distinguishes the EAN-128 code from Code 128.

Multiple AIs (along with their data) can be combined into a single bar code.

EAN128FNC1 is a UCC/EAN-128 (EAN128) type barcode that allows you to insert FNC1 character at any place and adjust the bar size, etc., which is not available in UCC/EAN-128.

To insert FNC1 character, set “\n” for C#, or “vbLf” for VB to Text property at runtime.

IntelligentMail

Intelligent Mail, formerly known as the 4-State Customer Barcode, is a 65-bar code used for domestic mail in the U.S.

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.

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.

PostNet

PostNet uses only numbers with a check digit.

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.

RSS14Stacked

RSS14Stacked uses the EAN.UCC information with Indicator digits as in the RSS14Truncated, but stacked in two rows for a smaller width.

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 runtime.
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 runtime.
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.
UCCEAN128	UCC/EAN –128 uses the complete ASCII character Set. This is a special version of Code 128 used in HIBC applications.
UPC_A	UPC-A uses only numbers (11 numbers and a check digit).
UPC_Eo	UPC-Eo 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_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.

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.

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.

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.

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: This property is not used by ActiveReports, but you can use it to store the path or relative path to an RPX report file that you want to load into a generic report instance in code.

Format

Textbox 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.

OleObject

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\ComponentOne\ActiveReports Developer 7\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.v7.dll, or for 64-bit machines, the GrapeCity.ActiveReports.Interop64.v7.dll.

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.

ChartControl

In ActiveReports, you can use the **ChartControl** 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 ChartControl 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 ChartControl 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 ChartControl.
- **Save As** allows you to save the current chart to an XML file that you can load into a ChartControl 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 ChartControl onto a report.
- **Data Source** opens the Chart Data Source dialog where you can build a connection string and create a query.

ReportInfo

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.

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

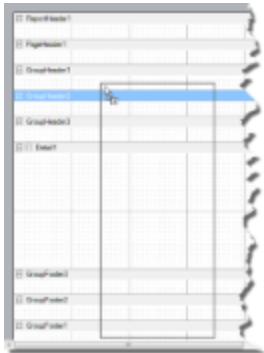


The CrossSectionLine control draws a vertical line from a header section to the corresponding footer section. At runtime, this vertical line stretches through any intervening sections. You can change the appearance of CrossSectionLine by changing the following properties.

Property Description

LineColor	Allows you to get or set color of the line.
LineStyle	Allows you to select the line style from solid, dash or dotted.
LineWeight	Allows you to specify the thickness of the line in pixel units.

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 in pixel units.
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.

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.

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, or DashDotDot.

Line weight: Enter the width in pixels 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.

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, or DashDotDot.

Line weight: Enter the width in pixels 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.

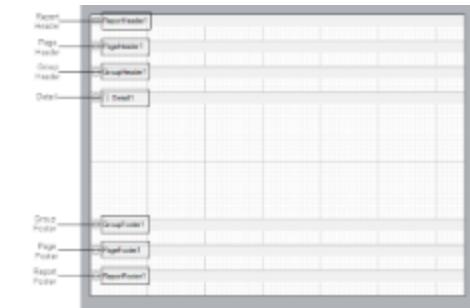
Radius: Enter the number of pixels for the amount of rounding to use on the corners of the box. Valid values are from 0 to 1000 pixels.

Section Report Structure

By default, a section report comprises 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 Developer 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 **Grouping Data in Section Reports**.

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 **Grouping Data in Section Reports**.

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.

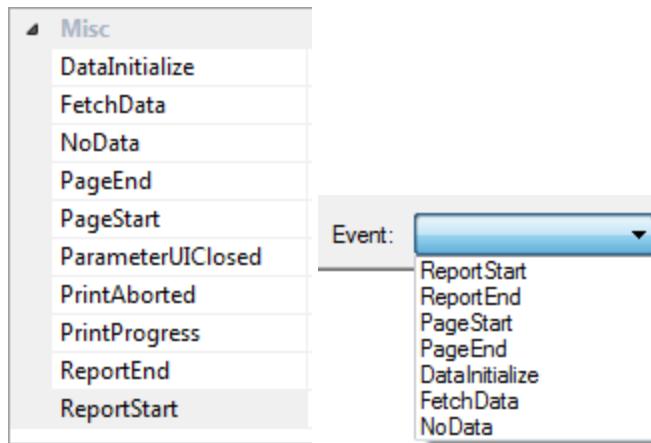
Event Sequence

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.

Scripting in Section Reports

In a section report, ActiveReports Developer 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 runtime 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.



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 Developer, 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 runtime 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.

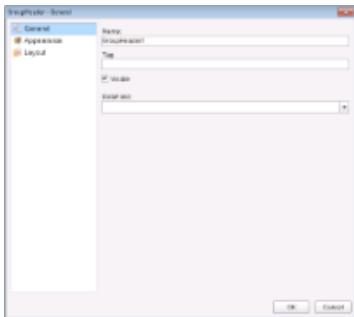
Grouping Data in Section Reports

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. See **Add Grouping in Section Reports** for further information on how to group data.

 **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.

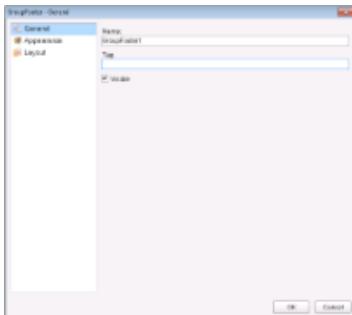
Appearance

- **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 runtime 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.

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)
- MMMMM = 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

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.
- **Releasing Reports** To properly release a report instance from memory, take these steps in the following order:
 1. Call the Dispose() method of the Document object
 2. Call the Dispose() method of the Report object
 3. Set the Report object to null

The code for properly releasing a report is as follows.

To release a report in Visual Basic

C# code.

```
rpt.Document.Dispose()  
rpt.Dispose()  
rpt = Nothing
```

To release a report in C#

C# code.

```
rpt.Document.Dispose();  
rpt.Dispose();  
rpt = null;
```

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.

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 its 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 or section report in **PDF** and **TIFF** formats. In page reports, it is also supported while rendering a report in Word, HTML, PDF and Image formats using rendering extensions. See **Rendering** for more information on rendering extensions.

Multiline in Report Controls

ActiveReports Developer 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

In a page 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 Developer, 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.



C H A R A C T E R
S P A C I N G

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

- 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, it is also supported while rendering a report through rendering extensions in Word, HTML, PDF and Image formats. See **Rendering** for further information on rendering extensions.

Exporting

ActiveReports Developer provides two independent ways to export a report to different formats:

- **Rendering Extensions.** You can use the Render method in Rendering Extensions of the PageDocument class to render a page report to Image, Html, Pdf, Xml and Word formats.
- **Export Filters.** You can use the Export method of the corresponding ExportFilter class to export a section report and a page report.

The following table illustrates the different export formats for section and page reports.

Export formats	Section report	Page 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.	✓	✗
Doc: Export reports to Word, a format that opens in Microsoft Word.	✗	✓
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, EMF, GIF, JPEG, or PNG image formats.	✗	✓
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 (Excel 2007).	✓	✓

Xml: Export reports to XML, a format that opens in a Web browser or delivers data to other applications.



When exporting a report to PDF with the **PDF Export** or the PDF rendering extension, you can use the following function.

- **Font Linking**

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**
- **PDF Export**
- **Text Export**
- **RTF Export**
- **TIFF Export**
- **Excel Export**

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 assembly in your project, or by selecting the object in the toolbox and adding it to the component tray below the Form which automatically adds this assembly to the project:

GrapeCity.ActiveReports.Export.Html.v7.dll

Note: HTML Export requires the .NET Framework full profile version. To ensure you are using the full profile version, go to the Visual Studio **Project** menu > **Properties** > **Compile** tab > **Advanced Compile Options** (for Visual Basic projects) or to the **Project** menu > **Properties** > **Application** tab (for C# projects) and under **Target framework** select a full profile version.

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
- CrossSectionBox control
- Control borders
- Shapes (other than filled rectangles)

LegacyHtml (HTML)

Usage:

- Create archival reports
- Open in Web browsers

Does not support:

- Line control
- Control borders
- Shapes (other than filled rectangles)

- CrossSectionBox and CrossSectionLine controls
- Overlapping controls

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 assembly in your project, or by selecting the object in the toolbox and adding it to the component tray below the Form which automatically adds this assembly to the project:

GrapeCity.ActiveReports.Export.Pdf.v7.dll

With the PDF export filter, you can use such features as **Font Linking**, Custom Font Factory and end-user defined characters.

 **Note:** These 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 and .emf). 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-line documentation)	A valid PdfSignature object	This must be set up in code. For more information, see Digital Signature Sample .

Version ('Version Property' in the on-line documentation)	Pdf11, 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. For a general overview of the PDF formats, see http://partners.adobe.com/public/developer/tips/topic_tip31.html For a detailed overview of PdfA1a, PdfA1b, PdfA2a, PdfA2b, and PdfA2u, see http://blogs.adobe.com/acrolaw/files/2011/05/PDFA_eSeminar_Slides_updated.pdf
--	--	---

PDF (Portable Document Format)

Usage:

- Create printable reports whose formats do not change from machine to machine.
- Open in Adobe Reader.

Does not support:

- All controls are supported.
- Dash and dot border patterns appear to look longer in the PDF output than in the ActiveReports Window Forms Viewer.

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.

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 ('DisplayMode Property' in the on-line documentation)	None (default), Outlines, Thumbs, or FullScreen	Select how to display bookmarks when the document is first opened. <ul style="list-style-type: none"> • 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 ('HideWindowUI Property' in the on-line documentation)	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 documentation)	String	Enter a subject to display in the Adobe Document Properties dialog, Description tab, Subject field.
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.

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 assembly in your project, or by selecting the object in the toolbox and adding it to the component tray below the Form which automatically adds this assembly to the project:

- GrapeCity.ActiveReports.Export.Xml.v7.dll

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.
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 assembly in your project, or by selecting the object in the toolbox and adding it to the component tray below the Form which automatically adds this assembly to the project:

- GrapeCity.ActiveReports.Export.Word.v7.dll

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 assembly in your project, or by selecting the object in the toolbox and adding it to the component tray below the Form which automatically adds this assembly to the project:

- GrapeCity.ActiveReports.Export.Excel.v7.dll

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 ('MultiSheet Property' in the on-line documentation)	True or False (default)	Set to True to export each page of your report to a separate sheet within the Excel 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 assembly in your project, or by selecting the object in the toolbox and adding it to the component tray below the Form which automatically adds this assembly to the project:

- GrapeCity.ActiveReports.Export.Image.v7.dll

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: <ul style="list-style-type: none"> • None delivers color output with no compression.

		<ul style="list-style-type: none"> • 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.

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.

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 Developer allows you to set parameters in your report to filter the data displayed. Parameters make navigation of the report easier for the user at runtime. See **Parameters** for further details.

Filters

The filtering feature is only available with page layout reports. By using filters in your page 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 page 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 have collation set i.e. reports that have two or more themes.

Parameters

ActiveReports Developer allows you to use parameters to filter or add the data to display in reports at runtime. You can either prompt users for parameters so that they control the output, or supply the parameters behind the scenes.

Page Report

In a page report, each parameter must be entered in three places: the Query page of the DataSet dialog, the Parameters page of the DataSet dialog, and the Report Parameters dialog for filtering data at runtime.

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 IN (?)
```

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.

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
```

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
 - **DateTime:** Presents the user with a calendar picker if 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.
- **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.

If you want to run a report without prompting the user for a value at runtime, 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 **Subreport in a CPL Report** 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 runtime, 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 runtime:
`<%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 runtime.
- 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.

 **Tip:** 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%>
```

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 Developer 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.

Type: This value, which defaults to S for string, tells ActiveReports Developer 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 Developer 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 control 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 Developer sends it to SQL in that format.

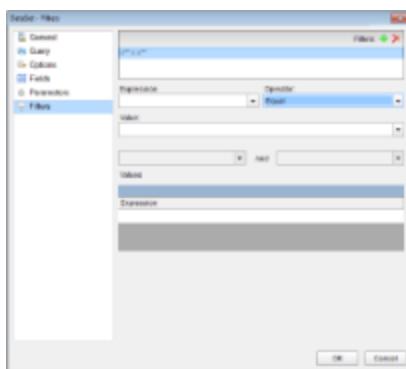
PromptUser: This Boolean allows you to tell ActiveReports Developer whether to prompt the user for a value. This can be set to True for some parameters and False for others. If you set the report's ShowParameterUI property to False, users are not prompted for any parameters, regardless of the PromptUser 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 in Page Reports** 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 Developer 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

Hyperlinks

Hyperlinks take you to a web page that opens in the default browser of the system. You can set hyperlinks in the Textbox and Image controls to access a Web page from your report. See **Add Hyperlinks** for further information.

Hyperlinks are displayed when you preview a page 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 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 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 and as a bookmarks panel for a section report. It

provides a convenient way to navigate a lengthy report.

Page Reports

In a page 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.

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.

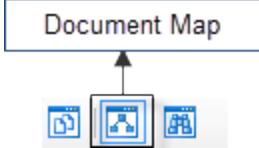
See **Add Bookmarks** for more information.

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

HTML

Effect on Document Map

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, 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.

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

In a page report, you can sort data in the data region, along with grouping, on a fixed page in an FPL 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 Developer 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 **Matrix**, **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 Fixed Page Layout (FPL), 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

In a page layout, 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 **AnnotationToolbarVisible Property (on-line documentation)** 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



AnnotationLine



can also point the balloon's tail in any direction using its **Quadrant** property.

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

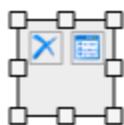


A rectangle with a background image and text. You can select an any image inside it using the **BackgroundImage** property. You can also place text on the image using the **Text** property.

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 [Using the Viewer](#) for more information.

1. In your Visual Studio project, on the Form where the Viewer control is placed, select the Viewer control and right-click to choose Properties.
2. In the **Properties Window** that appears, set the AnnotationToolbarVisible 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.



Note: You can print a page 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.

Windows Forms Viewer Customization

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

The ToolStrip 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.

Tool Click 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 [Viewer.ViewerToolbar Class](#), the **Customize the Viewer Control** topic, and the Custom Preview sample that is located in ...\\Documents\\ComponentOne Samples\\ActiveReports Developer 7\\Section Reports\\C#\\CustomPreview.

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.

Shrink Text to Fit in a Control

In ActiveReports Developer, when working with the Textbox control in a page report or a TextBox or Label control in a section report, you can use the **ShrinkToFit** property to reduce the size of the text so that it fits within the bounds of the control. The text shrinks at runtime, 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
Titanic	1997	8.1
Star Wars	1977	8
Die Hard	1988	7.1
Die Hard With a Vengeance	1995	7.3
Star Wars Episode I - The Phantom Menace	1999	8.1
Spider-Man	2002	8.6
Spider-Man 2	2004	8.5
Spider-Man 3	2007	8.5
Spider-Man 4	2009	8.6
Transformers	2007	8.1
Transformers: Revenge of the Fallen	2009	8.5
Transformers: Dark of the Moon	2011	8.6
Transformers: Age of Extinction	2014	8.6
Transformers: The Last Knight	2017	8.6
Jurassic Park	1993	8.6
Jurassic World	2015	8.6
Finding Nemo	2003	8.2
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.

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 using rendering extensions, ShrinkToFit is not supported in XML. However, all other rendering extensions allow ShrinkToFit display as it is. See **Rendering** for more on rendering extensions.

Standalone Designer and Viewer

ActiveReports Developer provides executable files for the Designer and the Viewer controls in the startup menu. These executable files function as standalone applications to help create, edit and 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 Standalone Viewer application to view the report.

To access the Standalone Designer or Viewer application

From the Start Menu, go to All Programs > ComponentOne > ActiveReports Developer and select **ActiveReports**

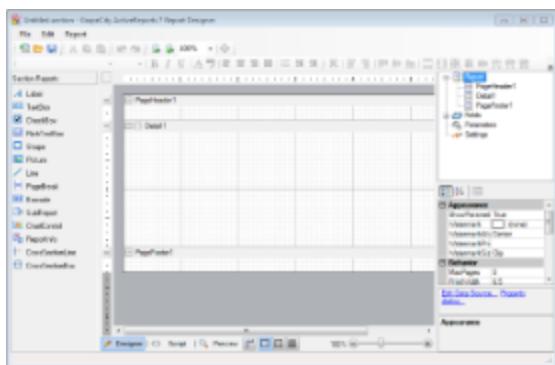
Developer Designer or ActiveReports Developer Viewer.

OR

Select the following applications located under ...\\Common Files\\ComponentOne\\ActiveReports Developer 7.

- GrapeCity.ActiveReports.Designer.exe
- GrapeCity.ActiveReports.Viewer.exe

Standalone Designer



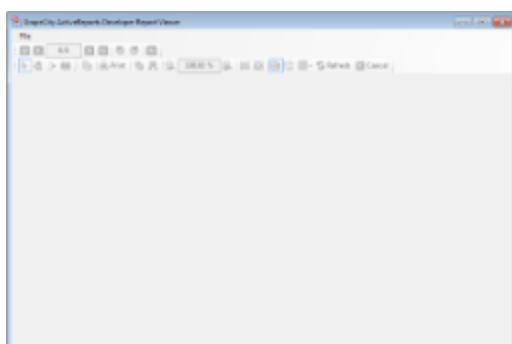
Standalone Designer refers to the GrapeCity.ActiveReports.Designer.exe bundled with the ActiveReports Developer installer. This application provides a user interface comprising of a Designer at the center along with a toolbox, toolbar, menu, Report Explorer and Properties Window to mimic the Visual Studio look and feel.

The Standalone Designer supports both section and page report layouts. By default, a standalone designer appears with a section layout loaded in the designer. To open a page layout, do one of the following:

- From the File menu > New menu option, open the **Create New Report** dialog and select Page Report.
- On the toolbar, select the New menu option to open the **Create New Report** dialog and select Page Report.

Use the File menu > **Save As** option to save the report in .rpx or .rdlx format depending on the type of layout you are using. The Report menu that appears in the menu bar is similar to the one in Visual Studio. See **Report Menu** for more information.

Standalone Viewer



Standalone Viewer refers to the GrapeCity.ActiveReports.Viewer.exe bundled with the ActiveReports Developer installer. This is basically a Windows Form application with an ActiveReports Developer Viewer control in it. The default user interface of this application provides an ActiveReports Developer Viewer control along with a menu bar.

You can open an .rdlx or .rpx report in the Standalone 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 standalone application.

Please note that any additional features activated through code like the annotation toolbar, are not available in the Standalone Viewer application. See **Using the Viewer** for more information on how to implement these features in the Viewer control.

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.v7.dll and the spoke is GrapeCity.ActiveReports.Viewer.Win.v7.resources.dll.

In your Program Files folder, the Localization folder is in a path like\ComponentOne\ActiveReports Developer 7\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. (The Flash viewer does not have a .bat file.)
- 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.v7.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.v7.resources.dll file to [ComponentOne](#) support 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 runtime 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)	ox007F	Invariant culture
af	ox0036	Afrikaans
af-ZA	ox0436	Afrikaans (South Africa)
sq	ox001C	Albanian

sq-AL	ox041C	Albanian (Albania)
ar	ox0001	Arabic
ar-DZ	ox1401	Arabic (Algeria)
ar-BH	ox3C01	Arabic (Bahrain)
ar-EG	oxoCo1	Arabic (Egypt)
ar-IQ	ox0801	Arabic (Iraq)
ar-JO	ox2Co1	Arabic (Jordan)
ar-KW	ox3401	Arabic (Kuwait)
ar-LB	ox3001	Arabic (Lebanon)
ar-LY	ox1001	Arabic (Libya)
ar-MA	ox1801	Arabic (Morocco)
ar-OM	ox2001	Arabic (Oman)
ar-QA	ox4001	Arabic (Qatar)
ar-SA	ox0401	Arabic (Saudi Arabia)
ar-SY	ox2801	Arabic (Syria)
ar-TN	ox1Co1	Arabic (Tunisia)
ar-AE	ox3801	Arabic (U.A.E.)
ar-YE	ox2401	Arabic (Yemen)
hy	ox002B	Armenian
hy-AM	ox042B	Armenian (Armenia)
az	ox002C	Azeri
az-Cyrl-AZ	ox082C	Azeri (Azerbaijan, Cyrillic)
az-Latn-AZ	ox042C	Azeri (Azerbaijan, Latin)
eu	ox002D	Basque
eu-ES	ox042D	Basque (Basque)
be	ox0023	Belarusian
be-BY	ox0423	Belarusian (Belarus)
bg	ox0002	Bulgarian
bg-BG	ox0402	Bulgarian (Bulgaria)
ca	ox0003	Catalan
ca-ES	ox0403	Catalan (Catalan)
zh-HK	oxoCo4	Chinese (Hong Kong SAR, PRC)
zh-MO	ox1404	Chinese (Macao SAR)
zh-CN	ox0804	Chinese (PRC)
zh-Hans	ox0004	Chinese (Simplified)
zh-SG	ox1004	Chinese (Singapore)
zh-TW	ox0404	Chinese (Taiwan)

zh-Hant	ox7C04	Chinese (Traditional)
hr	ox001A	Croatian
hr-HR	ox041A	Croatian (Croatia)
cs	ox0005	Czech
cs-CZ	ox0405	Czech (Czech Republic)
da	ox0006	Danish
da-DK	ox0406	Danish (Denmark)
dv	ox0065	Divehi
dv-MV	ox0465	Divehi (Maldives)
nl	ox0013	Dutch
nl-BE	ox0813	Dutch (Belgium)
nl-NL	ox0413	Dutch (Netherlands)
en	ox0009	English
en-AU	ox0C09	English (Australia)
en-BZ	ox2809	English (Belize)
en-CA	ox1009	English (Canada)
en-029	ox2409	English (Caribbean)
en-IE	ox1809	English (Ireland)
en-JM	ox2009	English (Jamaica)
en-NZ	ox1409	English (New Zealand)
en-PH	ox3409	English (Philippines)
en-ZA	ox1C09	English (South Africa)
en-TT	ox2C09	English (Trinidad and Tobago)
en-GB	ox0809	English (United Kingdom)
en-US	ox0409	English (United States)
en-ZW	ox3009	English (Zimbabwe)
et	ox0025	Estonian
et-EE	ox0425	Estonian (Estonia)
fo	ox0038	Faroese
fo-FO	ox0438	Faroese (Faroe Islands)
fa	ox0029	Farsi
fa-IR	ox0429	Farsi (Iran)
fi	ox000B	Finnish
fi-FI	ox040B	Finnish (Finland)
fr	ox000C	French
fr-BE	ox080C	French (Belgium)
fr-CA	ox0C0C	French (Canada)

fr-FR	ox040C	French (France)
fr-LU	ox140C	French (Luxembourg)
fr-MC	ox180C	French (Monaco)
fr-CH	ox100C	French (Switzerland)
gl	ox0056	Galician
gl-ES	ox0456	Galician (Spain)
ka	ox0037	Georgian
ka-GE	ox0437	Georgian (Georgia)
de	ox0007	German
de-AT	ox0C07	German (Austria)
de-DE	ox0407	German (Germany)
de-LI	ox1407	German (Liechtenstein)
de-LU	ox1007	German (Luxembourg)
de-CH	ox0807	German (Switzerland)
el	ox0008	Greek
el-GR	ox0408	Greek (Greece)
gu	ox0047	Gujarati
gu-IN	ox0447	Gujarati (India)
he	ox000D	Hebrew
he-IL	ox040D	Hebrew (Israel)
hi	ox0039	Hindi
hi-IN	ox0439	Hindi (India)
hu	ox000E	Hungarian
hu-HU	ox040E	Hungarian (Hungary)
is	ox000F	Icelandic
is-IS	ox040F	Icelandic (Iceland)
id	ox0021	Indonesian
id-ID	ox0421	Indonesian (Indonesia)
it	ox0010	Italian
it-IT	ox0410	Italian (Italy)
it-CH	ox0810	Italian (Switzerland)
ja	ox0011	Japanese
ja-JP	ox0411	Japanese (Japan)
kn	ox004B	Kannada
kn-IN	ox044B	Kannada (India)
kk	ox003F	Kazakh
kk-KZ	ox043F	Kazakh (Kazakhstan)

kok	ox0057	Konkani
kok-IN	ox0457	Konkani (India)
ko	ox0012	Korean
ko-KR	ox0412	Korean (Korea)
ky	ox0040	Kyrgyz
ky-KG	ox0440	Kyrgyz (Kyrgyzstan)
lv	ox0026	Latvian
lv-LV	ox0426	Latvian (Latvia)
lt	ox0027	Lithuanian
lt-LT	ox0427	Lithuanian (Lithuania)
mk	ox002F	Macedonian
mk-MK	ox042F	Macedonian (Macedonia, FYROM)
ms	ox003E	Malay
ms-BN	ox083E	Malay (Brunei Darussalam)
ms-MY	ox043E	Malay (Malaysia)
mr	ox004E	Marathi
mr-IN	ox044E	Marathi (India)
mn	ox0050	Mongolian
mn-MN	ox0450	Mongolian (Mongolia)
no	ox0014	Norwegian
nb-NO	ox0414	Norwegian (Bokmål, Norway)
nn-NO	ox0814	Norwegian (Nynorsk, Norway)
pl	ox0015	Polish
pl-PL	ox0415	Polish (Poland)
pt	ox0016	Portuguese
pt-BR	ox0416	Portuguese (Brazil)
pt-PT	ox0816	Portuguese (Portugal)
pa	ox0046	Punjabi
pa-IN	ox0446	Punjabi (India)
ro	ox0018	Romanian
ro-RO	ox0418	Romanian (Romania)
ru	ox0019	Russian
ru-RU	ox0419	Russian (Russia)
sa	ox004F	Sanskrit
sa-IN	ox044F	Sanskrit (India)
sr-Cyrl-CS	ox0C1A	Serbian (Serbia, Cyrillic)
sr-Latn-CS	ox081A	Serbian (Serbia, Latin)

sk	ox001B	Slovak
sk-SK	ox041B	Slovak (Slovakia)
sl	ox0024	Slovenian
sl-SI	ox0424	Slovenian (Slovenia)
es	ox000A	Spanish
es-AR	ox2CoA	Spanish (Argentina)
es-BO	ox400A	Spanish (Bolivia)
es-CL	ox340A	Spanish (Chile)
es-CO	ox240A	Spanish (Colombia)
es-CR	ox140A	Spanish (Costa Rica)
es-DO	ox1CoA	Spanish (Dominican Republic)
es-EC	ox300A	Spanish (Ecuador)
es-SV	ox440A	Spanish (El Salvador)
es-GT	ox100A	Spanish (Guatemala)
es-HN	ox480A	Spanish (Honduras)
es-MX	ox080A	Spanish (Mexico)
es-NI	ox4CoA	Spanish (Nicaragua)
es-PA	ox180A	Spanish (Panama)
es-PY	ox3CoA	Spanish (Paraguay)
es-PE	ox280A	Spanish (Peru)
es-PR	ox500A	Spanish (Puerto Rico)
es-ES	oxoCoA	Spanish (Spain)
es-ES_tradnl	ox040A	Spanish (Spain, Traditional Sort)
es-UY	ox380A	Spanish (Uruguay)
es-VE	ox200A	Spanish (Venezuela)
sw	ox0041	Swahili
sw-KE	ox0441	Swahili (Kenya)
sv	ox001D	Swedish
sv-FI	ox081D	Swedish (Finland)
sv-SE	ox041D	Swedish (Sweden)
syr	ox005A	Syriac
syr-SY	ox045A	Syriac (Syria)
ta	ox0049	Tamil
ta-IN	ox0449	Tamil (India)
tt	ox0044	Tatar
tt-RU	ox0444	Tatar (Russia)
te	ox004A	Telugu

te-IN	ox044A	Telugu (India)
th	ox001E	Thai
th-TH	ox041E	Thai (Thailand)
tr	ox001F	Turkish
tr-TR	ox041F	Turkish (Turkey)
uk	ox0022	Ukrainian
uk-UA	ox0422	Ukrainian (Ukraine)
ur	ox0020	Urdu
ur-PK	ox0420	Urdu (Pakistan)
uz	ox0043	Uzbek
uz-Cyril-UZ	ox0843	Uzbek (Uzbekistan, Cyrillic)
uz-Latn-UZ	ox0443	Uzbek (Uzbekistan, Latin)
vi	ox002A	Vietnamese
vi-VN	ox042A	Vietnamese (Vietnam)

How To

Learn to perform common tasks with ActiveReports with quick how-to topics.

This section contains information about

Page Report How To

Learn how to work with Page reports, both CPL and FPL, 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.

Customize, Localize, and Deploy

Learn how to customize each of the Windows and Web controls, how to localize reports and controls, and how to deploy Windows, Web, and Silverlight applications.

Use Fields in Reports

Learn how to utilize bound or database fields and calculated fields in reports.

Use Advanced Printing Options

Learn how to access and set up the advanced printing options provided with the ActiveReports Viewer.

Provide One-Touch Printing in the WebViewer (Pro Edition)

Learn how to use One-Touch printing in the Web Viewer.

Provide PDF Printing in the Silverlight Viewer (Pro Edition)

Learn how to use PDF printing in the Silverlight Viewer.

Print Methods In ActiveReports Developer

Learn about various Print methods in ActiveReports Developer.

Page Report How To

Learn to perform common tasks with ActiveReports with quick how-to topics.

This section contains information about how to

Work with Data

Learn how to connect to different data sources and add a data set in a page report.

Work with Report Controls and Data Regions

Learn how to use controls like the TextBox or Image, and data regions like the Matrix or Table on Page Reports.

Create Common Page Reports

Learn how to work with Page reports, both CPL and FPL, and perform various page report tasks.

Add Parameters in a Page Report

Learn how to add parameters to your report so that users can filter data.

Create and Add Themes

Learn how to create and add new themes.

Customize and Apply a Theme

Learn how to modify an existing theme and apply them in your report.

Use Constant Expressions in a Theme

Learn how to set a constant expression in a theme and display it on your report.

Set Up Collation

Learn how a report renders when multiple themes are applied.

Add Hyperlinks

Learn how to add hyperlinks and improve interactivity in your report.

Add Bookmarks

Learn how to add bookmark links and improve interactivity in your report.

Create and Use a Master Report

Learn how to create a master report and apply common features to any number of reports.

Export a Page Report (Export Filter)

Learn how to use Export Filters to export your report in HTML, PDF, RTF, Text, TIFF and Excel file formats.

Export a Page Report (Rendering Extension)

Learn how to use Rendering Extensions to render your report in Image, HTML, PDF, XML or Word file formats.

Sort Data

Learn how to sort data in page layout.

Allow Users to Sort Data in the Viewer

Learn how to use the Interactive Sort settings on a TextBox report control.

Create a Drill-Down Report

Learn how to use the Visibility settings to toggle detail data within a report.

Set a Drill-Through Link

Learn how to set a drill-through link.

Add Items to the Document Map

Learn how to add items in the Document Map.

Change Page Size

Learn how to change page size in page layout.

Add Page Breaks in CPL

Learn how to add page breaks in CPL reports.

Add Totals and Subtotals in a Data Region

Learn how to set totals and subtotals in all the data regions with illustrative examples.

Add Static Rows and Columns to a Matrix

Learn how to static rows and columns automatically or manually in a Matrix.

Cell Merging In Matrix

Learn how to cells with duplicate values are merged in a Matrix.

Work with Data

See step-by-step instructions for performing common tasks in Page Reports.

This section contains information about how to:

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 page report.

Create and Edit a Shared Data Source

Learn how to create and edit a shared data source.

Bind Reports to a Data Source at run time

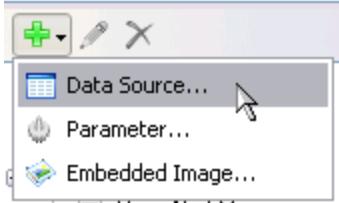
Learn how to bind a page report to a data source at run time.

Connect to a Data Source

In a page 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 Layout template in a Visual Studio project. See **Adding an ActiveReport to a Project** further information.

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 Data Source1. 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, OleDb, or Oracle as the data source Type value, Connection Properties, Connection String and Advanced Settings pages appear under Connection. See **Report Data Source Dialog** for further details.

Note: When you select an XML Provider under **Type**, the connection string must include XmlDocument or XmlData. Use of TransformationDoc is optional. See details of each of these connection string types below:

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.

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.

In ActiveReports, you can connect to most data sources with these steps. However, you need to follow the instructions below to connect to a Shared Data Source.

Connect to a Shared Data Source

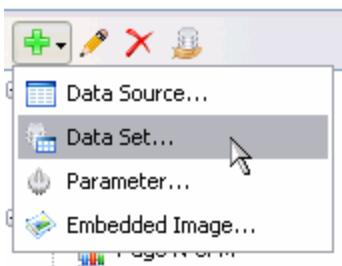
1. In the **Report Explorer**, right-click the Data Sources node and select the **Add Data Source** option or select **Data Source** from the Add button.
2. In the **Report Data Source** dialog that appears, select the **General** page and enter the name of the data source. This name appears as a child node to the Data Sources node in the Report Explorer.
3. Check the Shared Reference checkbox on.
4. Under Reference, click the **Browse** button and from 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.
5. Click the **OK** button on the lower right corner to close the dialog. A shared data source node appears in the Report Explorer.

Add a Dataset

In a page 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.

These steps assume that you have already added a Page Report template and connected it to a data source. See **Adding an ActiveReport to a Project** 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**.
 - If the **Command Type** is **Text** enter a SQL command or XML path in the **Query** box.
 - If the **Command Type** is **StoredProcedure** enter the name of the stored procedure in the **Query** box.
 - If the **Command Type** is **TableDirect** 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 data source provider, you have to provide an XML path on the Query page and generate fields using XPath expressions on the Fields page of the DataSet dialog. See the following example for details.

Query and Field settings for XML Data

The XML provider supports the use of XPath 1.0 in building queries and selecting Fields. Following is an example of how the query string and fields are set with an XMLData connection string.

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>;
```

Query String

Set the query like `/people/person/name`

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

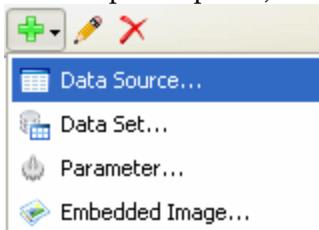
See [Use Fields in Reports](#) to understand fields further.

Create and Edit a Shared Data Source

You can save a data connection type such as an OleDb 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 template to create a new report. See [Adding an ActiveReport to a Project](#) 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 ProviderUnder 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 edit a shared data source

These steps assume that you have already connected your report to a shared data source. See [Connect to a Data Source](#) for further information.

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 Developer allows you to modify a data source at run time. See the following set of sample codes to connect your page report to a data source at run time.

OleDb Data Source

To connect to an OleDb data source at run time

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 **Adding an ActiveReport to a Project** and **Using the Viewer** for further information.

 **Note:** You can use the code example below for the SQL, Odbc, OleDb or Oracle 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 Documents
folder]\ComponentOne\ActiveReports Developer 7\Samples\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 Documents
folder]\ComponentOne\ActiveReports Developer 7\Samples\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.

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 runtime, 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 **Adding an ActiveReport to a Project** and **Using the 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 7 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 Documents folder]\ComponentOne\ActiveReports
Developer 7\Samples\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 Documents folder]\ComponentOne\ActiveReports
Developer 7\Samples\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 is included in the ActiveReports installation and is located in the *[UserDocumentFolder]\ComponentOne Samples\ActiveReports Developer 7\Page Reports\CPL\API* folder.

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 **Adding an ActiveReport to a Project** and **Using the 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 7 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.

Work with Report Controls and Data Regions

Learn to perform common tasks with ActiveReports with quick how-to topics.

This section contains information about how to

Grouping in a FixedPage

Learn how group data on the fixed page to apply grouping on the entire report instead of a single control.

Grouping in a Data Region

Learn how to group data in a data region like Table, List, BandedList, Matrix and Chart.

Set Detail Grouping In Sparklines

Learn how to set detail grouping to reflect a trend in your data with sparklines.

Set Filters in Page Reports

Learn how set filters in your report and limit the data you want to display.

Grouping in a FixedPage

In a fixed page layout, 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 **Adding an ActiveReport to a Project** and **Connect to a Data Source** for further information.

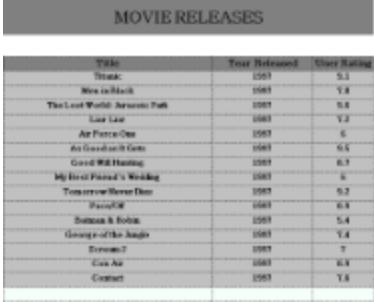
 **Note:** This topic uses the **Movie** table in the Reels database. By default, the Reels.mdb file is located in [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data folder.

To set grouping on a Fixed Page

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.



MOVIE RELEASES		
Title	Year Released	User Rating
Tron	1982	5.3
Home on the Range	1999	7.8
The Last World War	1999	9.8
Lady Macbeth	1960	7.2
Air Force One	1997	6
An Officer and a Gentleman	1982	6.5
Good Will Hunting	1997	8.7
My Best Friend's Wedding	1998	6
Tomorrow Never Dies	1997	5.2
Patriot	1995	6.8
Sabrina & Robin	1999	5.4
George of the Jungle	1997	7.4
Erin Brockovich	1999	7
Cosmopolis	1998	6.8
Conquest	1997	7.8

Grouping in a Data Region

In a page report, group your data from a data region on fields and expressions. Each data region provides grouping in a different way.

Use the steps below to learn how to set groups in each data region. These steps assume that you have already added a Page Report template and connected it to a data source. See **Adding an ActiveReport to a Project**, **Connect to a Data Source** and **Add a Dataset** for more information.

 **Note:** This topic uses examples from the Movie table in the Reels database. By default, in ActiveReports, the Reels.mdb file is located at [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data\Reels.mdb.

To set grouping in a Table

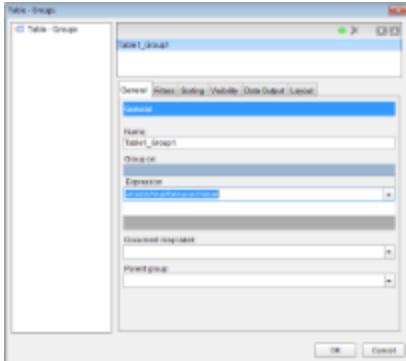
To group data in a **Table** data region, add group header and footer rows to the table or set detail grouping.

To add groups in a Table

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.

3. In the table dialog that appears go to the Groups page and click the Add (+) button to add a group manually.
4. In this dialog, on the General tab, under **Group on** enter the field name or expression on which you want to group the data. For e.g., `=Fields!YearReleased.Value`.



Note: A default group name like Table1_Group1 appears under **Name**, until you set the grouping expression under **Group On**, after which you can modify the group name.

5. 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.
6. Under **Parent group**, you can optionally set the parent group for a recursive hierarchy.
7. Click **OK** to close the dialog. Group header and group footer rows appear right above and below the table details row.
8. Drag and drop fields onto the Table data region and in the group header row, add the field on which the grouping is set.
9. Preview the report to see the result.

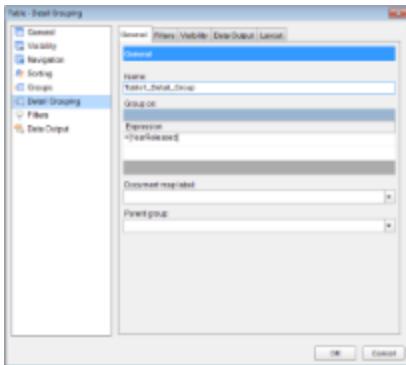
Movie Database		
	Year	User Rating
Mr. Smith Goes to Washington	1939	NR
African Queen	1951	NR
Rebel Without a Cause	1955	NR
From Here to Eternity	1953	NR
Seven Year Itch	1955	NR
Brave One	1956	NR
George Cukor's Casablanca	1942	NR
Good Will Hunting	1997	NR
North by Northwest	1959	NR
The Hustle	1975	NR
What Ever Happened to Baby Jane?	1962	NR
Seven	1950	NR
Conceited Movie Star	1955	NR
1960		
Cat Ballou	1965	NR
One Flew Over the Cuckoo's Nest	1975	NR
Smash Up and Fly	1960	NR
The Hunt	1963	NR
1970		
One Flew Over the Cuckoo's Nest	1975	NR
Death Wish	1974	NR
Death Wish II	1982	NR
The Hunt	1973	NR
1980		
Die Hard	1988	NR
Die Hard With a Vengeance	1995	NR
Die Hard: A Christmas Carol	1995	NR
Die Hard: With a Vengeance	1998	NR
Die Hard: Special Edition	1998	NR
Lethal Weapon 3: A Sudden Death	1992	NR

Tip: You can sort, filter, set page breaks or repeat headers for your grouped data in the other tabs of the Table-Groups dialog.

To set detail grouping in a Table

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.

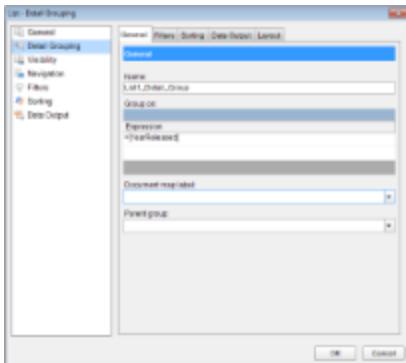


 **Tip:** You can filter, set page breaks or set visibility for your grouped data in the other tabs of the Table-Detail Grouping page.

To set detail grouping in a List

To group data in a **List** data region, set detail grouping. 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 List data region onto the design surface.
2. With the List selected on the report, under the Properties window, click the **Property dialog** link to open the List 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 List1_Detail_Group appears under **Name**, until you set the grouping expression under **Group On**, after which you can modify the group name.

- Under the **Document map label** field, you can optionally set a label to add the item to the document map. See [Document Map](#) for further information.
 - Under the **Parent group** field, you can optionally set the parent group for a recursive hierarchy.
 - Click **OK** to close the dialog.
 - Drag and drop fields or other data regions onto the List data region and go to the preview tab to see grouped data.

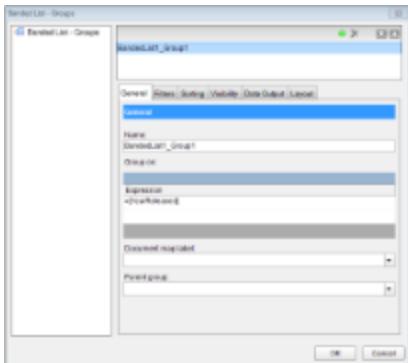


 Tip: You can place lists within lists to create nested grouping. You can also filter, set page breaks or set sorting for your grouped data in the other tabs of the List-Detail Grouping page.

To set grouping in a BandedList

To group data in a **BandedList** data region, add the BandedList group header and footer row.

1. From the Visual Studio toolbox, drag and drop a BandedList data region onto the design surface.
 2. Right-click the BandedList data region and select **Insert Group** to open the Banded List-Groups dialog.
OR
With the BandedList selected on the report, under the Properties Window, click the **Property dialog** link to open the Banded List dialog > Groups page. However, you have to click the Add (+) button and add the group manually in this case.
 3. In the Groups page, under **Group on** enter the field name or expression on which you want to group the data.
For e.g., `=Fields!YearReleased.Value`.



 Note: A default group name like BandedList1_Group1 appears under **Name**, until you set the grouping expression under **Group On**, after which you can modify the group name.

- Under the **Document map label** field, you can optionally set a label to add the item to the document map. See **Document Map** for further information.
 - Under **Parent group** you can set the parent group for a recursive hierarchy.
 - Click the **OK** to close the dialog.
 - Drag and drop fields onto the BandedList data region and go to the preview tab to see grouped data.



To set grouping in a Matrix

To group data in a **Matrix** data region, set row and column groups dynamically or manually in the data region.

To dynamically group a matrix using dynamic columns and rows

1. From the Visual Studio toolbox, drag and drop a Matrix data region onto the design surface.
 2. From the **Report Explorer**, drag a field and drop it onto a row or column header in the matrix. This action adds both an expression to the textbox in the cell and a grouping to the matrix.

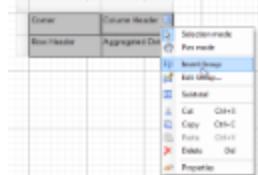
Corner	Column Header
Row Header	Aggregated Detail

3. From the **Report Explorer**, drag and drop a field onto the aggregate detail cell of the matrix data region.

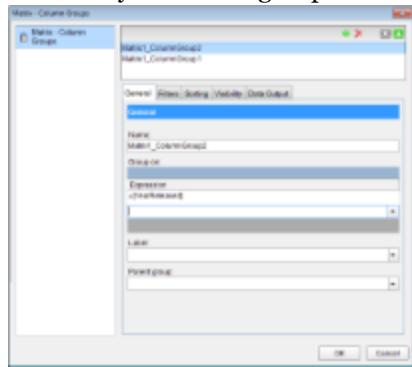
To manually group a matrix using dynamic columns and rows

1. From the Visual Studio toolbox, drag and drop a Matrix data region onto the design surface.
 2. Right-click a row or column header and choose **Insert Group**. This adds a new cell below the original column

header or to the right of the original row header.



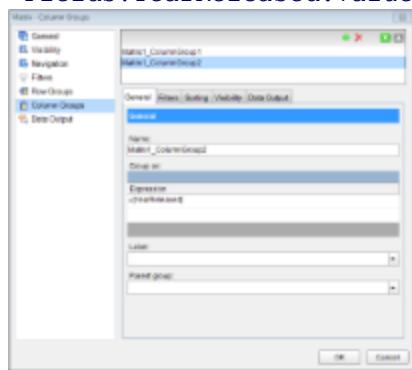
3. In the **Matrix - Column Groups** or **Row Groups** dialog that appears, under **Group on** enter an expression on which you want to group the data. For e.g., `=Fields!YearReleased.Value`.



4. Click **OK** to close the dialog.
5. Drag and drop or set fields in the aggregate detail cell of the matrix data region and go to the **Preview Tab** to view the result.

Or

1. From the Visual Studio toolbox, drag and drop a **Matrix** data region onto the design surface.
2. With the Matrix data region selected on the report, under the Properties Window, click the **Property dialog** link to open the Matrix dialog.
3. In the **Column Groups** or **Row Groups** page, click the **Add** button to add a new group and a new cell below the original column header or to the right of the original row header.
4. Enter an expression under **Group on** on which you want to group the data. For e.g., `=Fields!YearReleased.Value`.



5. Click **OK** to close the dialog.
6. Drag and drop or set fields in the aggregate detail cell of the matrix data region and go to the **Preview Tab** to view the result.

Note: A default group name like Matrix1_RowGroup1 or Matrix1_ColumnGroup1 appears under **Name**, until you set the grouping expression under **Group On**, after which you can modify the group name.

To set grouping in a Chart

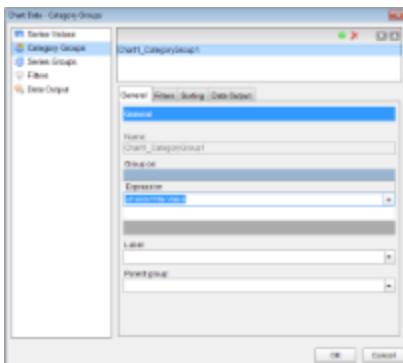
To group data in a **Chart** data region, set grouping for categories or series. In a Chart, you can set groups directly on the design surface or add category groups and series groups manually.

To dynamically group a Chart category or series

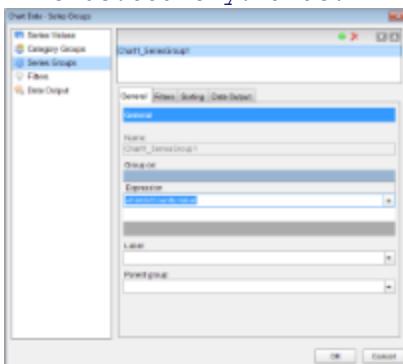
1. From the Visual Studio toolbox, drag and drop a Chart data region onto the design surface.
2. From the **Report Explorer**,
3. Drag a field near the lower edge of the Chart and drop it onto the part that reads **Drop category fields here**. This action groups the categorized chart data on the field set here.
4. Drag a field near the upper edge of the Chart and drop it onto the part that reads **Drop data fields here**. This adds data which is to be grouped to the chart.
5. Drag a field near the right edge of the Chart and drop it onto the part that reads **Optionally drop series fields here**. This action groups data on series.

To manually group a Chart category or series

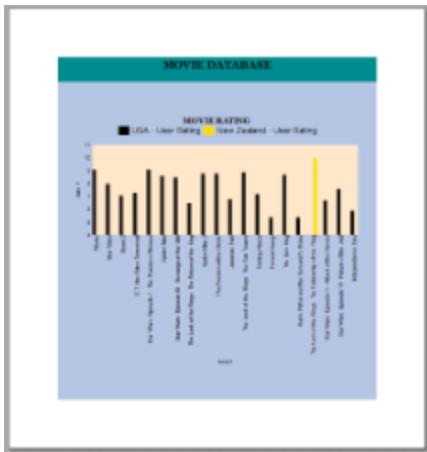
1. From the Visual Studio toolbox, drag and drop a **Chart** data region onto the design surface.
2. With the Chart data region selected on the report, under the Properties Window, click the **Chart Data** link to open the Chart Data dialog.
3. In the Chart Data dialog, go to the Category Groups page and click the **Add** button to add a new group.
4. Enter an expression under **Group on** on which you want to categorize the data. For e.g.,
`=Fields!Title.Value`.



5. In the Chart Data dialog, go to the Series Groups page and click the **Add** button to add a new group.
6. Enter an expression under **Group on** on which you want to group your series data. For e.g.,
`=Fields!Country.Value`.



7. Click **OK** to close the dialog.
8. Drag the **UserRating** field near the upper edge of the Chart and drop it onto the part that reads, **Drop data fields here**. This adds data which is to be grouped to the chart.



 **Note:** A default group name like Chart1_CategoryGroup1 or Chart1_SeriesGroup1 appear under **Name**, until you set the grouping expression under **Group On**, after which you can modify the group name.

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 template to your project, connected it to a data source and added a dataset. See **Adding an ActiveReport to a Project**, **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. By default, in ActiveReports Developer, the Reels.mdb file is located in the [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data folder.

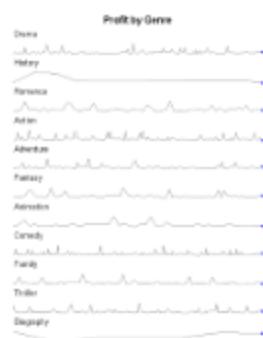
1. From the toolbox, drag a **Table** data region onto the report design surface.
2. With the table selected, set the following:
 - Set the **DataSetName** property (For example, SalesByGenre).
 - Right-click the table handle to the left of the Detail row and select **Insert Row Below** or **Insert Row 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.



Set Filters in Page Reports

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

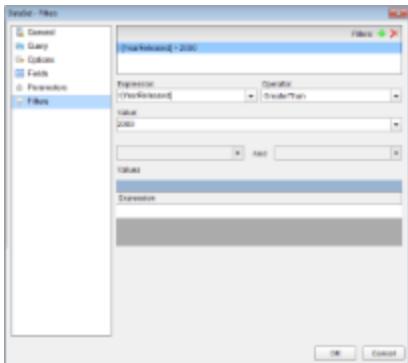
In a fixed page layout (FPL), you can also set filters on the page through the FixedPage dialog.

Use the following steps to create filters in a page report. These steps assume that you have already added a Page Report template and have a data connection and a dataset in place. See **Adding an ActiveReport to a Project**, **Connect to a Data Source** and **Add a Dataset** for more information.

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

MOVIE RELEASES POST YEAR 2000			
Title	Year Released	User Rating	
The Princess Diaries	2001	PG	
Elizabethtown	2005	G	
Knocked Up	2007	G	
Knockout	2008	G	
Open Season	2008	G	
Planet of the Apes	2001	G	
A Beautiful Mind	2001	G	
The Constant Gardener	2005	G	
Harry Potter and the Goblet of Fire	2005	G	
Monsters, Inc.	2001	G	
South Park: Bigger, Longer & Uncut	2006	G	
Monsters vs. Aliens	2009	G	
Adventures of Tintin	2011	G	
Shane	2009	G	
Just Watch Your Step	2007	G	
The Bourne Identity	2002	G	
The Bourne Supremacy	2004	G	
Death Proof	2007	G	
Grindhouse	2007	G	
The Lord of the Rings: The Fellowship of the Ring	2001	G	
Heavy Metal and the Chemistry of Success	2007	G	
The Last Castle	2007	G	
Monsters vs. Aliens	2009	G	
Old Dogs	2009	G	
Open Water	2009	G	
Reindeer Games	2009	G	
My Big Fat Greek Wedding	2004	G	

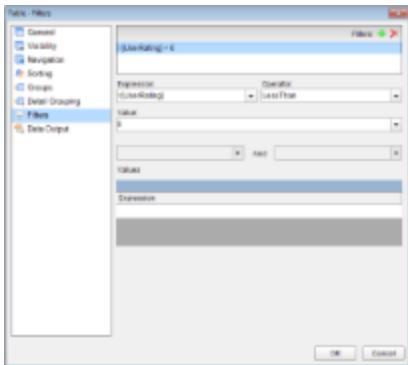
Set filters in a Data Region

When you set a filter in a data region, you can limit the amount of data available for use inside that data region.

- 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 **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.



- 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`

WORST MOVIES IN PAST YEARS		
Title	Year Released	User Rating
One Flew Over the Cuckoo's Nest	1975	5.7
The Godfather	1972	9.3
Love Story	1970	9.3
The Godfather	1971	9.3
Breaking Away	1979	5.3
One Flew Over the Cuckoo's Nest	1975	5.7
Citizen Kane	1941	9.3
New Moon: Eclipse II - The Source to New Moon	1960	5.3
Alien	1979	8
Reindeer of the Loch Ness	1967	5.7
One Flew Over the Cuckoo's Nest	1975	5.7
The Godfather	1971	9.3
One Flew Over the Cuckoo's Nest	1975	5.7
Coming Home	1978	8.1
One Flew Over the Cuckoo's Nest	1975	5.7
Who Framed Roger Rabbit	1982	8.2
One Flew Over the Cuckoo's Nest	1975	5.7
Driving Miss Daisy	1989	9.3
The Godfather	1971	9
Empire State	1989	8.1
Alien	1979	8.7
One Flew Over the Cuckoo's Nest	1975	5.7
Forrest Gump	1994	9.3
One Flew Over the Cuckoo's Nest	1975	5.7
Empire State	1989	8.1
Predator	1987	8.9
Gremlins	1984	8.3

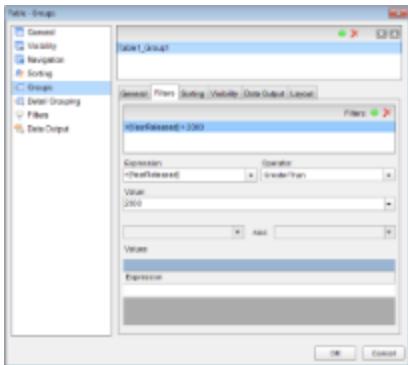
Set filters on groups in a Data Region

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.

- In the page report, set grouping on a data region. For example, on a Table data region, set grouping on the `=Fields!YearReleased.Value` field. See **Grouping in a Data Region** for further details.
- 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 **Chart** data region, right-click the data region and choose the **Chart data** option to open the Chart Data dialog.

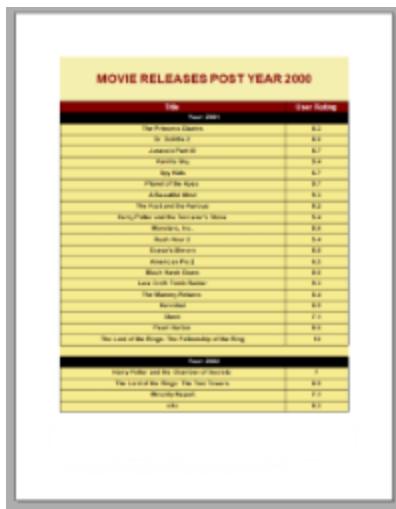
- In the Table dialog, go to the **Groups** tab and select the Group.
- After selecting the group, go to the **Filters** tab and click the Add (+) icon to add a new filter. By default, an empty filter expression gets added to the filter list.
- Under Expression, enter an expression or use the Expression Editor to provide the expression on which to filter data. For example, `=Fields!YearReleased.Value`
- Under Operator, select the 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.



- 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`



Create Common Page Reports

See step-by-step instructions for creating commonly used reports in a Page Layout.

This section contains information about how to:

Create Top N Report

Learn how to display top N data on a report.

Create Red Negatives Report

Learn to highlight negative values in red on a report.

Create Green Bar Report

Learn to create alternate background colors for report details.

Create a Bullet Graph

Learn how to create a Bullet Graph.

Create a Whisker Sparkline

Learn how to create a Whisker Sparkline.

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 template to your project and connected it to a data source. See **Adding an ActiveReport to a Project** 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 FPL 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:



TOP 10 MOVIES OF ALL TIME		
Title	Year	IMDB Rating
The Lord of the Rings: The Fellowship of the Ring	2001	8.9
The Lord of the Rings: The Return of the King	2003	8.9
Lethal Weapon 3	1992	8.8
The Empire Strikes Back	1980	8.7
The Godfather	1972	8.7
The Dark Knight	2008	8.6
Star Wars: Episode V - The Empire Strikes Back	1980	8.5
Seven Samurai	1954	8.5
Seven Samurai	1954	8.5
The Silence of the Lambs	1991	8.3
Gladiator	2000	8.3
Raiders of the Lost Ark	1981	8.3
Brave	2012	8.2

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 template to your project, connected it to a data source and added a DataSet. See **Adding an ActiveReport to a Project**, **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:

Accounts Data		
Description	Balance	Date Of Approval
Accounts	0	
Accounts	0	
Net Assets	1	
Current Assets	1	
Current Assets	0	
Capital & Reserves - RMB	0	
Capital & Reserves - RMB	0	
Current Liabilities	1	
Current Liabilities	1	
Service & Administration	1	2012-09-01 00:00:00
Information Systems	1	2012-09-01 00:00:00
Marketing	1	2012-09-01 00:00:00
Cost	1	2012-09-01 00:00:00
Service & Administration	1	2012-09-01 00:00:00
Information Systems	1	2012-09-01 00:00:00
Marketing	1	2012-09-01 00:00:00
Cost	1	2012-09-01 00:00:00
Service & Administration	1	2012-09-01 00:00:00
Information Systems	1	2012-09-01 00:00:00
Marketing	1	2012-09-01 00:00:00
Cost	1	2012-09-01 00:00:00
Service & Administration	1	2012-09-01 00:00:00
Information Systems	1	2012-09-01 00:00:00
Marketing	1	2012-09-01 00:00:00
Cost	1	2012-09-01 00:00:00
Service & Administration	1	2012-09-01 00:00:00
Information Systems	1	2012-09-01 00:00:00
Marketing	1	2012-09-01 00:00:00
Cost	1	2012-09-01 00:00:00
Service & Administration	1	2012-09-01 00:00:00
Information Systems	1	2012-09-01 00:00:00
Marketing	1	2012-09-01 00:00:00
Cost	1	2012-09-01 00:00:00

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 template to your project, connected it to a data source and added a dataset. See [Adding an ActiveReport to a Project](#), [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:

Title	No Stock	Market Price
Movie White and Black	0	10.00
Movie with No Stock	0	10.00
One Thousand and One Nights	1	10.00
New Products	0	10.00
Movie Images	1	10.00
The Jungle Book	0	10.00
Book History of Chinese Culture	1	10.00
Love Story	0	10.00
The Godfather	1	10.00
The King	0	10.00
The Training Officer	0	10.00
Planning Officer	0	10.00
...more	1	0
One Thousand and One Nights	0	10.00
Book	0	10.00
Book Encyclopedia of Chinese	1	10.00
New Movie	1	10.00
Information Nightly	0	10.00
Something on the Should	0	0.00
...more	1	0.00
New Movie	0	10.00
Book	0	10.00
Book Encyclopedia	0	10.00
New Movie	0	10.00
Information Nightly	0	10.00
Something on the Should	0	0.00
...more	1	0.00
New Movie	0	10.00
Book	0	10.00
Book Encyclopedia	0	10.00
New Movie	0	10.00
Information Nightly	0	10.00
Something on the Should	0	0.00
...more	1	0.00

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 template to your project and connected it to a data source. See **Adding an ActiveReport to a Project** 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.



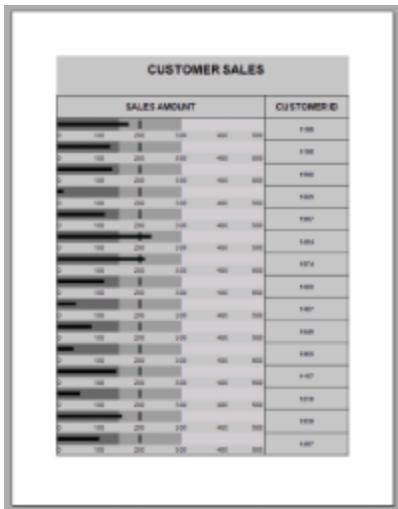
3. With the Bullet control selected on the design surface:

- 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.

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 template to your project, connected it to a data source and added a dataset. See **Adding an ActiveReport to a Project, Connect to a Data Source and Add a Dataset** for more information.

 **Note:** These steps use the AccountsChartData table from the Reels database. The sample Reels.mdb database file is located in the [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data folder.

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.



Add Parameters in a Page Report

You can add parameters to a page 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 valueSelect 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:

Create an ALL Parameter

Learn how to create a multi-value parameter with an ALL value.

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 be fetched using the parameter value without prompting the user.

Create an ALL Parameter

In a page report, when you create a **Multivalue** list in a parameter, you can add an ALL value which removes the parameter filter to this list. Set the following to create an ALL parameter:

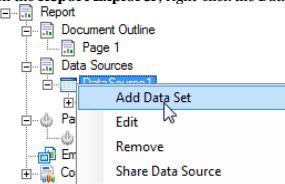
- A dataset to populate the parameter values
- A report parameter
- A list of values for the report parameter
- A dataset with a parameter

The following procedures take you through a step by step process of how to set an ALL parameter. These steps assume that you have added a page layout template to your report and have a data connection in place. See [Adding an ActiveReport to a Project and Connect to a Data Source](#) for further information.

 **Note:** This topic uses the Producers and Movie table from the Reels database. By default, in ActiveReports, the Reels.mdb file is located at [User Documents folder]\ComponentOne Samples\ActiveReports\Developer 7\Data\Reels.mdb.

To create a dataset to populate the parameter values

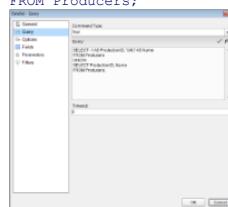
1. In the **Report Explorer**, right-click the Data Source (DataSource1 by default) node and select **Add Data Set**.



2. In the **DataSet** dialog that appears, select the **Query** page.

3. Enter a SQL query like the following into the **Query** text box where a UNION SELECT statement combines the results of the ALL query with those of the query for the individual values.

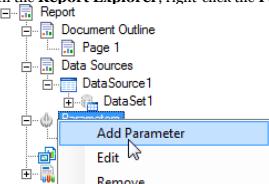
```
SELECT -1 AS ProductionID, " (All) " AS Name
FROM Producers
UNION
SELECT ProductionID, Name
FROM Producers;
```



4. Click the **OK** button on the lower right corner to close the dialog and see the dataset and the selected fields appear in the Report Explorer.

To add a Report Parameter

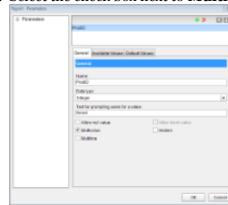
1. In the **Report Explorer**, right-click the **Parameters** node and select **Add Parameter**.



2. In the **Report - Parameters** dialog that appears, add a name for the parameter (i.e. ProdID) and ensure that the **Data type** matches that of the field (i.e. Integer for ProductionID).

3. Enter **Text for prompting users for a value**.

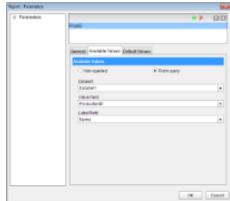
4. Select the check box next to **Multivalue** to allow users to select more than one item from the list.



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 above (i.e. DataSet1).

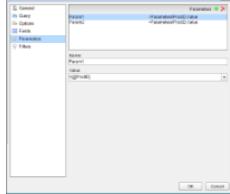
3. Under the **Value** field, select the field given for -1 (i.e. ProductionID).
4. Under **Label** field, select the field given for "All" (i.e. Name).
5. Click the **OK** button on the lower right corner to close the dialog and add the parameter to the collection.



To add a dataset with a parameter

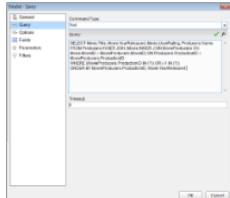
1. In the **Report Explorer**, right-click the Data Source (DataSource1 by default) node and select **Add Data Set**.
2. In the DataSet dialog that appears, on the **Parameters** page, click the Add (+) icon above the parameters list and add the following to the dataset to provide values for the parameters we add to the query in step 3 below.

Name: Param1; **Value:** =Parameters!ProdID.Value
Name: Param2; **Value:** =Parameters!ProdID.Value



3. On the **Query** page, enter a SQL query like the following in the **Query** text box:

```
SELECT Movie.Title, Movie.YearReleased, Movie.UserRating, Producers.Name
FROM Producers INNER JOIN (Movie INNER JOIN MovieProducers ON Movie.MovieID = MovieProducers.MovieID) ON Producers.ProductionID = MovieProducers.ProductionID
WHERE (MovieProducers.ProductionID IN (?) OR (-1 IN (?)))
ORDER BY MovieProducers.ProductionID, Movie.YearReleased
```

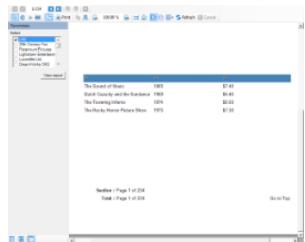


4. Click the **Validate DataSet** icon to validate the query and to populate the Fields list.



5. Click the **OK** button on the lower right corner to close the dialog and see the dataset and the selected fields appear in the Report Explorer.

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 an ALL option at the top.



Note: In a fixed page layout (FPL), 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.

Add a Cascading Parameter

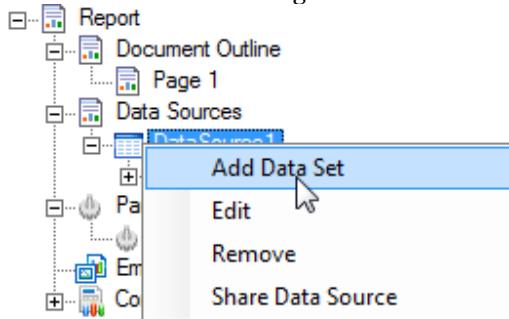
When a parameter's value list depends on the value of another parameter, the report collects the required parameter value and uses it to create the value list for the second parameter. This cascade of parameter values is sometimes also called dependent or hierarchical parameters.

Use the following instructions to create your own cascading parameters. These steps assume that you have added a page layout template to your report and have a data connection in place. See **Adding an ActiveReport to a Project** 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. By default, in ActiveReports, the Reels.mdb file is located at [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data\Reels.mdb.

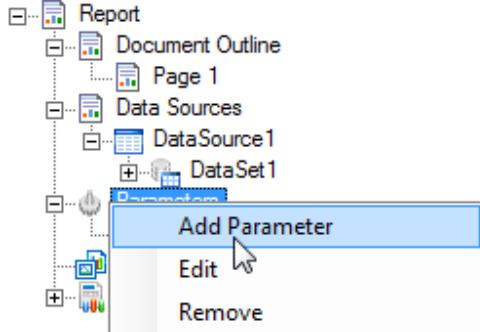
1. In the **Report Explorer**, right-click the Data Source (DataSource1 by default) node and select **Add Data Set** to

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.

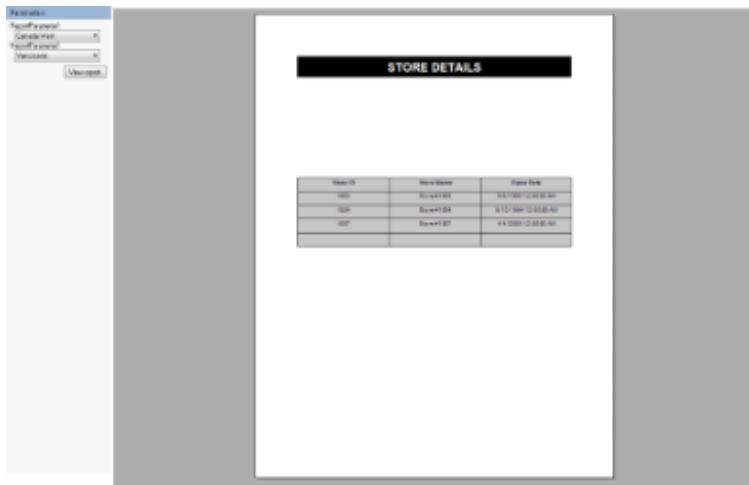
10. In the Report Explorer, right-click the **Parameters** node and select **Add Parameter**



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 fixed page layout (FPL), 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

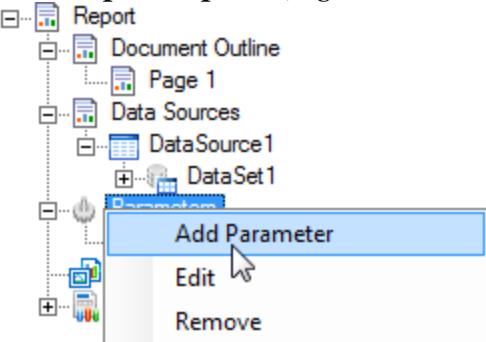
If you want to run a report without prompting the user for a value at runtime, you need to set a default value for each parameter. The report collects the required parameter value from the default value and uses it to generate the report.

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 layout template to your report and have a data connection in place. See **Adding an ActiveReport to a Project** 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. By default, in ActiveReports Developer, the Reels.mdb file is located at [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data folder.

1. In the **Report Explorer**, right-click the **Parameters** node and select **Add Parameter**.



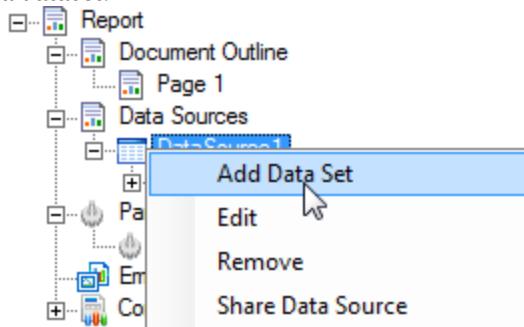
2. In the **Report - Parameters** dialog that appears, add a parameter named **StorePrice** with an Integer data type. Click the checkbox next to **Hidden** to hide the parameter UI at runtime.
3. On the Default Values tab, select **Non-queried** and click the Add(+) icon to add an empty expression for 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.

- 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.

- In the Report Explorer, right-click Data Source (DataSource1 by default) node and select **Add Data Set** to create a dataset.



- In the **DataSet Dialog** that appears, on the Parameters page, click the Add(+) icon to add an empty expression for the parameter.
- 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`
- 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 (?)`.
- 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.
- 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.



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 **CollateBy** (**'CollateBy Property' in the on-line documentation**) property to control the page order in a report. For more information, see **Collation**.

Use the following instructions to create and add themes.

To create a new theme

1. From the **Start** menu, go to **All Programs > ComponentOne > ActiveReports Developer** and select **ActiveReports Developer 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 a Theme

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 ('Themes Property' in the on-line documentation)** 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.

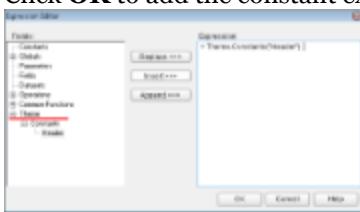
Use the following instructions to create and use constant expressions in **themes**.

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 the 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 **CollateBy ('CollateBy Property' in the on-line documentation)** property.

Caution: If you are using collation in a report, you cannot use interactive features, such as drill down, links, document map, and sorting.

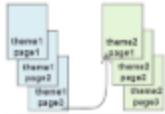
You can control the page order of a rendered report with multiple themes by selecting the collation mode in the **CollateBy ('CollateBy Property' in the on-line documentation)** property of the report:

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** ('**CollateBy Property**' in the on-line documentation) property and select one of the available options:

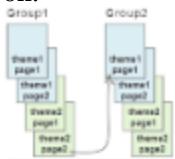
- **Simple.** Renders report pages without any specific sorting. For example, if you have a report with 2 themes, the report renders all pages with theme 1, then all pages with theme 2.



- **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 CPL reports, the **Value** collation mode is not available by design.

See **Add Page Numbering** for information on setting cumulative page count formats for FPL reports.

Add Hyperlinks

In a page report, you can set hyperlinks in the **TextBox** or **Image** controls to access a Web page from your report. 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,
<http://www.gcpowertools.com/>
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 PowerTools.

Add 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 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.

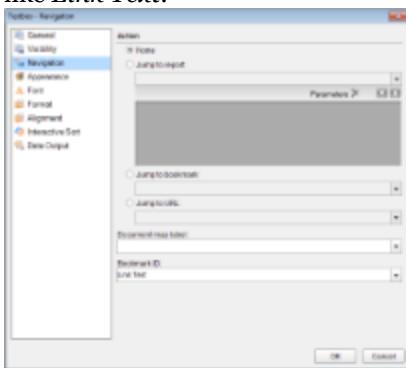
Use the following steps to create a Bookmark ID on a Textbox control which functions as the title of the report page and create a bookmark link on another Textbox control at the bottom of the page.

These steps assume that you have already added a Page Report template to your project. See **Adding an ActiveReport to a Project** 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*.



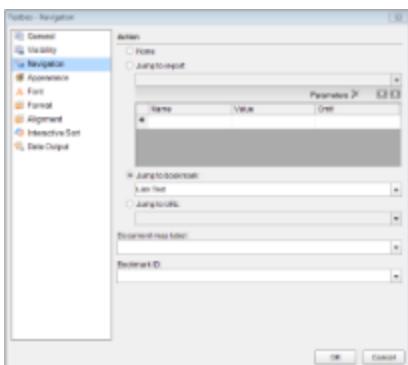
5. Click **OK** to close the dialog.

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. You can only set a bookmark link on a Textbox or an Image control.

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 and Use a Master Report

In a page report, you can create a master report that you apply 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

These steps assume that you have already added a Page Report template and connected it to a data source. As the Master Report feature is available in Continuous Page Layout (CPL), convert your report to this layout, in case you have a Fixed Page Layout (FPL) open.

See the topics, **Adding an ActiveReport to a Project** and **To Convert an FPL to CPL** in **Exploring Page Reports** 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**, then **Page Footer**.
3. From the toolbox, drag and drop controls in the page header and footer. These controls that appear on every page of every report to when you apply the master report. For e.g., an image of the company logo, or a textbox with the company Web site address.
4. From the toolbox, drag and drop a ContentPlaceHolder control to the report. This control defines where you want to allow controls to be placed on content reports that use the master report.

Add a ContentPlaceHolder

A ContentPlaceHolder control appears in the toolbox when you convert a CPL report to a Master Report. This

control provides the region to use for creating or designing a content report after applying a master report template. 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 the final output.

5. With the report selected, go to the Report menu > Save Layout.
6. 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 the rdlx-master file format.

To use a master report

These steps assume that you have already added a Page Report template and converted it to a CPL report. This report functions as a content report.

1. With focus on the report to which you want to apply the master report, from the Report menu, select **Set Master Report**.
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.

Export a Page Report (Export Filter)

In a page layout, ActiveReports provides various export filters that you can use to export reports to supported file formats. Here are the export formats that are included with ActiveReports:

- **HTML** For displaying on Web browsers or e-mail. You can access the HTML Export filter by adding the reference to *GrapeCity.ActiveReports.Export.Html.v7.dll* in your project.
- **PDF** For preserving formatting on different computers. You can access the PDF Export filter by adding the reference to *GrapeCity.ActiveReports.Export.Pdf.v7.dll* in your project.
- **RTF** 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.v7.dll* in your project.
- **Text** 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.v7.dll* in your project.
- **TIFF** For transmitting faxes. You can access the Image Export filter by adding the reference to *GrapeCity.ActiveReports.Export.Image.v7.dll* in your project.
- **Excel** For displaying as spreadsheets. You can access the Excel Export filter by adding the reference to *GrapeCity.ActiveReports.Export.Excel.v7.dll* in your project.

 **Note:** HTML Export requires the .NET Framework full profile version. To ensure you are using the full profile version, go to the Visual Studio **Project** menu > **Properties** > **Compile** tab > **Advanced Compile Options** (for Visual Basic projects) or to the **Project** menu > **Properties** > **Application** tab (for C# projects) and under **Target framework** select a full profile version.

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. See **Adding ActiveReports Controls** for further information.

To export a report

1. Place the **Invoice.rdlx** report inside your project's **Bin>Debug** folder.
2. In the Solution Explorer, right-click the References node and select **Add Reference**.
3. In the Add Reference dialog that appears, select the following references and click **OK** to add them to your project.
GrapeCity.ActiveReports.Export.Excel.v7
GrapeCity.ActiveReports.Export.Html.v7
GrapeCity.ActiveReports.Export.Image.v7
GrapeCity.ActiveReports.Export.Pdf.v7

```
GrapeCity.ActiveReports.Export.Word.v7  
GrapeCity.ActiveReports.Export.Xml.v7
```

4. Double-click the Windows Form to create a Form_Load event.
5. Add the following code to add Invoice.rdlx to your project.

To write the code in Visual Basic.NET

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

```
' Create a page report rpt.  
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 ("invoice.rdlx"))  
Dim MyDocument As New GrapeCity.ActiveReports.Document.PageDocument (rpt)
```

To write the code in C#

C# code. Paste INSIDE the Form_Load event.

```
// Create a page report rpt.  
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 ("invoice.rdlx"));  
GrapeCity.ActiveReports.Document.MyDocument = new GrapeCity.ActiveReports.Document.PageDocument (rpt);
```

6. Add the following code to export Invoice.rdlx to all the formats.

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.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 + "\\TIFFEExpt.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 + "\\XLSEExpt.xlsx");
```

-
7. Press **F5** to run the application. The exported files are saved in the bin\debug folder.

Export a Page Report (Rendering Extension)

You can use the rendering extensions for Image, HTML, PDF, XML or Word to render a page report in any of the supported formats. See **Rendering** for details on rendering formats.

The following steps provide an example of rendering a report in the format.

1. In Visual Studio, create a new Windows Forms 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.

To write the code in Visual Basic.NET

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

```
' Provide the page report you want to render.  
Dim _reportDef As New GrapeCity.ActiveReports.PageReport(New  
System.IO.FileInfo("C:\MovieRatings.rdlx"))  
Dim _reportRuntime As New GrapeCity.ActiveReports.Document.PageDocument(_reportDef)  
  
' Set the file format you want the report to be rendered in.  
Dim exportFile As String = System.IO.Path.GetTempFileName() + ".pdf"  
Dim myFile As New System.IO.FileInfo(exportFile)  
  
' Provide settings for your rendering output.  
Dim settings As New GrapeCity.ActiveReports.Export.Pdf.Page.Settings()  
settings.HideToolbar = True  
settings.HideMenubar = True  
settings.HideWindowUI = True  
  
' Set the rendering extension and render the report.  
Dim _renderingExtension As New  
GrapeCity.ActiveReports.Export.Pdf.Page.PdfRenderingExtension()  
Dim _provider As New  
GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider(myFile.Directory,  
System.IO.Path.GetFileNameWithoutExtension(myFile.Name))  
_reportRuntime.Render(_renderingExtension, _provider, settings)  
System.Diagnostics.Process.Start(exportFile)
```

To write the code in C#

C# code. Paste INSIDE the Form Load event.

```
// Provide the page report you want to render.  
GrapeCity.ActiveReports.PageReport _reportDef = new
```

```
GrapeCity.ActiveReports.PageReport(new System.IO.FileInfo(@"C:\MovieRatings.rdlx"));
GrapeCity.ActiveReports.Document.PageDocument _reportRuntime = new
GrapeCity.ActiveReports.Document.PageDocument(_reportDef);

// Set the file format you want the report to be rendered in.
string exportFile = System.IO.Path.GetTempFileName() + ".pdf";
System.IO.FileInfo myFile = new System.IO.FileInfo(exportFile);

// Provide settings for your rendering output.
GrapeCity.ActiveReports.Export.Pdf.Page.Settings settings = new
GrapeCity.ActiveReports.Export.Pdf.Page.Settings();
settings.HideToolbar = true;
settings.HideMenubar = true;
settings.HideWindowUI = true;

// Set the rendering extension and render the report.
GrapeCity.ActiveReports.Export.Pdf.Page.PdfRenderingExtension _renderingExtension = new
GrapeCity.ActiveReports.Export.Pdf.Page.PdfRenderingExtension();
GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider _provider = new
GrapeCity.ActiveReports.Rendering.IO.FileStreamProvider(myFile.Directory,
System.IO.Path.GetFileNameWithoutExtension(myFile.Name));
_reportRuntime.Render(_renderingExtension, _provider, settings);
System.Diagnostics.Process.Start(exportFile);
```

 **Note:** The code above works for any supported rendering format. Simply, replace the instances of PDF with the specific rendering format type and use the desired rendering extension.

Sort Data

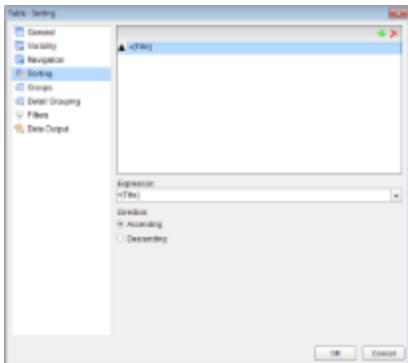
In a page report, you can apply sorting on a data region, grouped data or fixed page.

Use the following steps to determine how to sort your data. These steps assume that you have already added a Page Report (xml-based) template to your project and connected it to a data source. See **Adding an ActiveReport to a Project, Connect to a Data Source** and **Add a Dataset** for more information.

Sorting 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.

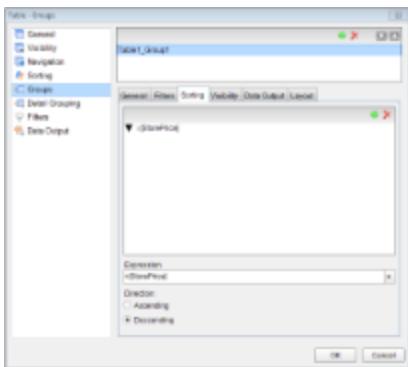
The following image shows the result after setting sorting in a **Table** data region on the **Title** field in Ascending order:

Sorting 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 **Grouping in a Data Region** 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.

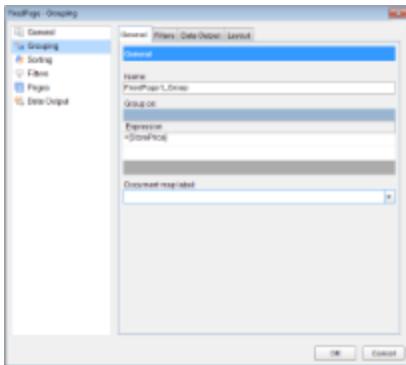
1. On the Groups or the Detail Grouping page of the data region dialog, select the **Sorting** tab.
 2. In the **Sorting** tab, click the Add(+) icon to add an empty expression to the sorting list.
 3. In the Expression field, enter the expression directly or use <Expression...> from the dropdown to 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.



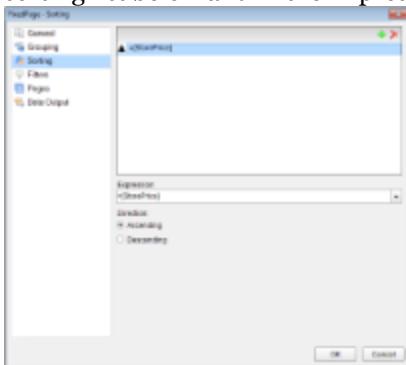
4. Under **Direction**, select Ascending or Descending to set the order in which to sort your data.
5. Click **OK** to close the dialog.
6. From the **Report Explorer**, drag and drop the field on which sorting is set and go to the Preview Tab to view the result.

The following image shows the result on a **Table** data region where grouping and sorting is set on the **StorePrice** field in Descending order:

DVD STOCK		
ID	On Stock	Store Price
1000	20	10.99
1001	20	10.99
1002	20	10.99
1003	20	10.99
1004	20	10.99
1005	20	10.99
1006	20	10.99
1007	20	10.99
1008	20	10.99
1009	20	10.99
1010	20	10.99
1011	20	10.99
1012	20	10.99
1013	20	10.99
1014	20	10.99
1015	20	10.99
1016	20	10.99
1017	20	10.99
1018	20	10.99
1019	20	10.99
1020	20	10.99
1021	20	10.99
1022	20	10.99
1023	20	10.99
1024	20	10.99
1025	20	10.99
1026	20	10.99
1027	20	10.99
1028	20	10.99
1029	20	10.99
1030	20	10.99
1031	20	10.99
1032	20	10.99
1033	20	10.99
1034	20	10.99
1035	20	10.99
1036	20	10.99
1037	20	10.99
1038	20	10.99
1039	20	10.99
1040	20	10.99
1041	20	10.99
1042	20	10.99
1043	20	10.99
1044	20	10.99
1045	20	10.99
1046	20	10.99
1047	20	10.99
1048	20	10.99
1049	20	10.99
1050	20	10.99
1051	20	10.99
1052	20	10.99
1053	20	10.99
1054	20	10.99
1055	20	10.99
1056	20	10.99
1057	20	10.99
1058	20	10.99
1059	20	10.99
1060	20	10.99
1061	20	10.99
1062	20	10.99
1063	20	10.99
1064	20	10.99
1065	20	10.99
1066	20	10.99
1067	20	10.99
1068	20	10.99
1069	20	10.99
1070	20	10.99
1071	20	10.99
1072	20	10.99
1073	20	10.99
1074	20	10.99
1075	20	10.99
1076	20	10.99
1077	20	10.99
1078	20	10.99
1079	20	10.99
1080	20	10.99
1081	20	10.99
1082	20	10.99
1083	20	10.99
1084	20	10.99
1085	20	10.99
1086	20	10.99
1087	20	10.99
1088	20	10.99
1089	20	10.99
1090	20	10.99
1091	20	10.99
1092	20	10.99
1093	20	10.99
1094	20	10.99
1095	20	10.99
1096	20	10.99
1097	20	10.99
1098	20	10.99
1099	20	10.99
1100	20	10.99
1101	20	10.99
1102	20	10.99
1103	20	10.99
1104	20	10.99
1105	20	10.99
1106	20	10.99
1107	20	10.99
1108	20	10.99
1109	20	10.99
1110	20	10.99
1111	20	10.99
1112	20	10.99
1113	20	10.99
1114	20	10.99
1115	20	10.99
1116	20	10.99
1117	20	10.99
1118	20	10.99
1119	20	10.99
1120	20	10.99
1121	20	10.99
1122	20	10.99
1123	20	10.99
1124	20	10.99
1125	20	10.99
1126	20	10.99
1127	20	10.99
1128	20	10.99
1129	20	10.99
1130	20	10.99
1131	20	10.99
1132	20	10.99
1133	20	10.99
1134	20	10.99
1135	20	10.99
1136	20	10.99
1137	20	10.99
1138	20	10.99
1139	20	10.99
1140	20	10.99
1141	20	10.99
1142	20	10.99
1143	20	10.99
1144	20	10.99
1145	20	10.99
1146	20	10.99
1147	20	10.99
1148	20	10.99
1149	20	10.99
1150	20	10.99
1151	20	10.99
1152	20	10.99
1153	20	10.99
1154	20	10.99
1155	20	10.99
1156	20	10.99
1157	20	10.99
1158	20	10.99
1159	20	10.99
1160	20	10.99
1161	20	10.99
1162	20	10.99
1163	20	10.99
1164	20	10.99
1165	20	10.99
1166	20	10.99
1167	20	10.99
1168	20	10.99
1169	20	10.99
1170	20	10.99
1171	20	10.99
1172	20	10.99
1173	20	10.99
1174	20	10.99
1175	20	10.99
1176	20	10.99
1177	20	10.99
1178	20	10.99
1179	20	10.99
1180	20	10.99
1181	20	10.99
1182	20	10.99
1183	20	10.99
1184	20	10.99
1185	20	10.99
1186	20	10.99
1187	20	10.99
1188	20	10.99
1189	20	10.99
1190	20	10.99
1191	20	10.99
1192	20	10.99
1193	20	10.99
1194	20	10.99
1195	20	10.99
1196	20	10.99
1197	20	10.99
1198	20	10.99
1199	20	10.99
1200	20	10.99
1201	20	10.99
1202	20	10.99
1203	20	10.99
1204	20	10.99
1205	20	10.99
1206	20	10.99
1207	20	10.99
1208	20	10.99
1209	20	10.99
1210	20	10.99
1211	20	10.99
1212	20	10.99
1213	20	10.99
1214	20	10.99
1215	20	10.99
1216	20	10.99
1217	20	10.99
1218	20	10.99
1219	20	10.99
1220	20	10.99
1221	20	10.99
1222	20	10.99
1223	20	10.99
1224	20	10.99
1225	20	10.99
1226	20	10.99
1227	20	10.99
1228	20	10.99
1229	20	10.99
1230	20	10.99
1231	20	10.99
1232	20	10.99
1233	20	10.99
1234	20	10.99
1235	20	10.99
1236	20	10.99
1237	20	10.99
1238	20	10.99
1239	20	10.99
1240	20	10.99
1241	20	10.99
1242	20	10.99
1243	20	10.99
1244	20	10.99
1245	20	10.99
1246	20	10.99
1247	20	10.99
1248	20	10.99
1249	20	10.99
1250	20	10.99
1251	20	10.99
1252	20	10.99
1253	20	10.99
1254	20	10.99
1255	20	10.99
1256	20	10.99
1257	20	10.99
1258	20	10.99
1259	20	10.99
1260	20	10.99
1261	20	10.99
1262	20	10.99
1263	20	10.99
1264	20	10.99
1265	20	10.99
1266	20	10.99
1267	20	10.99
1268	20	10.99
1269	20	10.99
1270	20	10.99
1271	20	10.99
1272	20	10.99
1273	20	10.99
1274	20	10.99
1275	20	10.99
1276	20	10.99
1277	20	10.99
1278	20	10.99
1279	20	10.99
1280	20	10.99
1281	20	10.99
1282	20	10.99
1283	20	10.99
1284	20	10.99
1285	20	10.99
1286	20	10.99
1287	20	10.99
1288	20	10.99
1289	20	10.99
1290	20	10.99
1291	20	10.99
1292	20	10.99
1293	20	10.99
1294	20	10.99
1295	20	10.99
1296	20	10.99
1297	20	10.99
1298	20	10.99
1299	20	10.99
1300	20	10.99
1301	20	10.99
1302	20	10.99
1303	20	10.99
1304	20	10.99
1305	20	10.99
1306	20	10.99
1307	20	10.99
1308	20	10.99
1309	20	10.99
1310	20	10.99
1311	20	10.99
1312	20	10.99
1313	20	10.99
1314	20	10.99
1315	20	10.99
1316	20	10.99
1317	20	10.99
1318	20	10.99
1319	20	10.99
1320	20	10.99
1321	20	10.99
1322	20	10.99
1323	20	10.99
1324	20	10.99
1325	20	10.99
1326	20	10.99
1327	20	10.99
1328	20	10.99
1329	20	10.99
1330	20	10.99
1331	20	10.99
1332	20	10.99
1333	20	10.99
1334	20	10.99
1335	20	10.99
1336	20	10.99
1337	20	10.99
1338	20	10.99
1339	20	10.99
1340	20	10.99
1341	20	10.99
1342	20	10.99
1343	20	10.99
1344	20	10.99
1345	20	10.99
1346	20	10.99
1347	20	10.99
1348	20	10.99
1349	20	10.99
1350	20	10.99
1351	20	10.99
1352	20	10.99
1353	20	10.99
1354	20	10.99
1355	20	10.99
1356	20	10.99



- 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.



- Under **Direction**, select Ascending or Descending to set the order in which to sort your data.
- Click **OK** to close and apply the settings.
- 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

DVDs for sorted: \$0			
Product ID	Title	As Stock	Store Price
1000	The King's Speech	0	\$8.00
1001	Drive	0	\$8.00
1002	Up In The Air	0	\$8.00
1003	Seven Years In Tibet	0	\$8.00
1004	Pearl Harbor	0	\$8.00
1005	Braveheart	0	\$8.00
1006	The Thin Red Line	0	\$8.00
1007	Unbroken	0	\$8.00
1008	Citizen Kane	0	\$8.00
1009	Seven Samurai	0	\$8.00
1010	Toy Story	0	\$8.00
1011	Law & Order	0	\$8.00
1012	Seven Weeks	0	\$8.00
1013	Highway To Hell	0	\$8.00
1014	Raging Bull	0	\$8.00
1015	Schindler's List	0	\$8.00
1016	Star Wars: Episode I - The Phantom Menace	0	\$8.00
1017	One Flew Over the Cuckoo's Nest	0	\$8.00
1018	Westworld	0	\$8.00
1019	Empire	0	\$8.00
1020	Braveheart	0	\$8.00
1021	Monty Python	0	\$8.00
1022	The Big Lebowski	0	\$8.00
1023	Citizen Kane	0	\$8.00
1024	The Return Of The King	0	\$8.00
1025	The Passion Of The Christ	0	\$8.00
1026	The Godfather	0	\$8.00
1027	The Silence Of The Lambs	0	\$8.00

Page 2

DVDs for sorted: \$0.00			
Product ID	Title	As Stock	Store Price
1000	The King's Speech	0	\$8.00
1001	Drive	0	\$8.00
1002	Up In The Air	0	\$8.00
1003	Seven Years In Tibet	0	\$8.00
1004	Pearl Harbor	0	\$8.00
1005	Braveheart	0	\$8.00
1006	The Thin Red Line	0	\$8.00
1007	Unbroken	0	\$8.00
1008	Citizen Kane	0	\$8.00
1009	Seven Samurai	0	\$8.00
1010	Toy Story	0	\$8.00
1011	Law & Order	0	\$8.00
1012	Seven Weeks	0	\$8.00
1013	Highway To Hell	0	\$8.00
1014	Raging Bull	0	\$8.00
1015	Schindler's List	0	\$8.00
1016	Star Wars: Episode I - The Phantom Menace	0	\$8.00
1017	One Flew Over the Cuckoo's Nest	0	\$8.00
1018	Westworld	0	\$8.00
1019	Empire	0	\$8.00
1020	Braveheart	0	\$8.00
1021	Monty Python	0	\$8.00
1022	The Big Lebowski	0	\$8.00
1023	Citizen Kane	0	\$8.00
1024	The Return Of The King	0	\$8.00
1025	The Passion Of The Christ	0	\$8.00
1026	The Godfather	0	\$8.00
1027	The Silence Of The Lambs	0	\$8.00

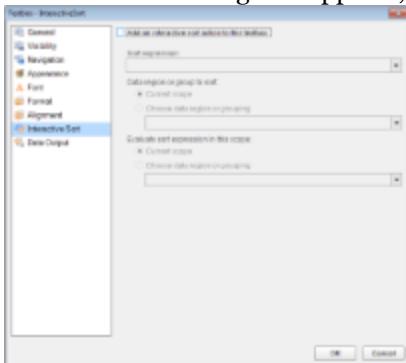
Allow Users to Sort Data in the Viewer

In a page 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) 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.

Title	Year Released	User Rating
The Godfather	1972	9.1
Star Wars	1977	8
Home Alone	1990	7.9
E.T. The Extra-Terrestrial	1982	7.7
One Flew Over the Cuckoo's Nest	1975	8.1
Spirited Away	2001	8.8
Star Wars Episode III: Revenge of the Sith	2005	8.3
Home Alone 2: Lost in New York	1992	6.5
Spirited Away	2001	8.8
The Passion of the Christ	2004	8.8
Forrest Gump	1994	8.8
The Last of the Mohicans	1992	8.9
Die Hard	1988	7.2
Forrest Gump	1994	8.8
The Lion King	1994	8.7
Home Alone and the Screenwriter's Strike	2001	8.4
The Lord of the Rings: The Return of the King	2003	10

You can click the icon to sort in descending order. The icon changes to an up arrow that you can click to sort ascending.



Title	Year Released	User Rating
The Star Wreck Project	1990	5
Bands & Bounces	1991	5
Die Hard 2	1990	5
Nine to Five	1980	5
Holmes on the Lost Ark	1981	5.5
Aladdin	1992	5.5
Reindeer Games	1998	5.5
Apollo 13	1995	5.5
The Waterboy	1998	5.5
On Another Day	2001	5.5
The Jungle Book	1967	5.5
One-Eyed Jacks (The Wild West)	1961	5.5
Coming to America	1988	5.5
Something's Gotta Give	2003	5.5
The Village	2004	5.5
Overboard II	1981	5.2
Star Wars: Episode V - The Empire Strikes Back	1980	5.5

Create a Drill-Down Report

In a page layout, you can set up data regions, report controls, table rows, and matrix 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 item**.
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.



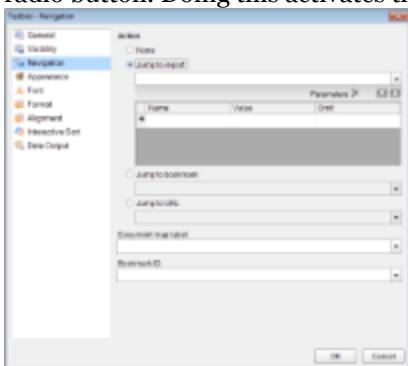
Click the icon to view the hidden data.



Set a Drill-Through Link

In a page report, you can use drill-through links to connect one report to another detailed report. Follow the steps to set a drill-through link in a page report:

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.



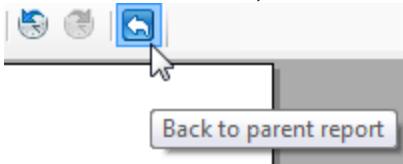
3. Under the **Jump to Report** field, enter the name of the report (like BasicReport.rdlx) that you want to navigate to on clicking the drill-through link.

 **Note:** Under the **Jump to Report** field, when you enter just a report name, the targeted report should be kept in the same directory as the parent report. Else, you can also enter the relative path to the location of the report. You can also use a report from your connected database using the Custom Resource Locator.

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



Title	Year	User Rating
Singin' in the Rain	1952	9.2
The Wizard of Oz	1939	9.2
Vertigo	1958	9.2
Rear Window	1954	9.2
The Long Goodbye	1973	9.1
The Godfather	1972	9.2
The Silence of the Lambs	1991	9.2
Seven Samurai	1954	9.2
The Lord of the Rings: The Fellowship of the Ring	2001	9.2
Seven	2002	9.1
Die Hard	1988	9.0
The Godfather	1974	9.2

View Top Movies Of
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
20100
20101
20102
20103
20104
20105
20106
20107
20108
20109
20110
20111
20112
20113
20114
20115
20116
20117
20118
20119
20120
20121
20122
20123
20124
20125
20126
20127
20128
20129
20130
20131
20132
20133
20134
20135
20136
20137
20138
20139
20140
20141
20142
20143
20144
20145
20146
20147
20148
20149
20150
20151
20152
20153
20154
20155
20156
20157
20158
20159
20160
20161
20162
20163
20164
20165
20166
20167
20168
20169
20170
20171
20172
20173
20174
20175
20176
20177
20178
20179
20180
20181
20182
20183
20184
20185
20186
20187
20188
20189
20190
20191
20192
20193
20194
20195
20196
20197
20198
20199
201000
201001
201002
201003
201004
201005
201006
201007
201008
201009
201010
201011
201012
201013
201014
201015
201016
201017
201018
201019
201020
201021
201022
201023
201024
201025
201026
201027
201028
201029
201030
201031
201032
201033
201034
201035
201036
201037
201038
201039
201040
201041
201042
201043
201044
201045
201046
201047
201048
201049
201050
201051
201052
201053
201054
201055
201056
201057
201058
201059
201060
201061
201062
201063
201064
201065
201066
201067
201068
201069
201070
201071
201072
201073
201074
201075
201076
201077
201078
201079
201080
201081
201082
201083
201084
201085
201086
201087
201088
201089
201090
201091
201092
201093
201094
201095
201096
201097
201098
201099
201100
201101
201102
201103
201104
201105
201106
201107
201108
201109
201110
201111
201112
201113
201114
201115
201116
201117
201118
201119
201120
201121
201122
201123
201124
201125
201126
201127
201128
201129
201130
201131
201132
201133
201134
201135
201136
201137
201138
201139
201140
201141
201142
201143
201144
201145
201146
201147
201148
201149
201150
201151
201152
201153
201154
201155
201156
201157
201158
201159
201160
201161
201162
201163
201164
201165
201166
201167
201168
201169
201170
201171
201172
201173
201174
201175
201176
201177
201178
201179
201180
201181
201182
201183
201184
201185
201186
201187
201188
201189
201190
201191
201192
201193
201194
201195
201196
201197
201198
201199
201200
201201
201202
201203
201204
201205
201206
201207
201208
201209
201210
201211
201212
201213
201214
201215
201216
201217
201218
201219
201220
201221
201222
201223
201224
201225
201226
201227
201228
201229
201230
201231
201232
201233
201234
201235
201236
201237
201238
201239
201240
201241
201242
201243
201244
201245
201246
201247
201248
201249
201250
201251
201252
201253
201254
201255
201256
201257
201258
201259
201260
201261
201262
201263
201264
201265
201266
201267
201268
201269
201270
201271
201272
201273
201274
201275
201276
201277
201278
201279
201280
201281
201282
201283
201284
201285
201286
201287
201288
201289
201290
201291
201292
201293
201294
201295
201296
201297
201298
201299
201300
201301
201302
201303
201304
201305
201306
201307
201308
201309
201310
201311
201312
201313
201314
201315
201316
201317
201318
201319
201320
201321
201322
201323
201324
201325
201326
201327
201328
201329
201330
201331
201332
201333
201334
201335
201336
201337
201338
201339
201340
201341
201342
201343
201344
201345
201346
201347
201348
201349
201350
201351
201352
201353
201354
201355
201356
201357
201358
201359
201360
201361
201362
201363
201364
201365
201366
201367
201368
201369
201370
201371
201372
201373
201374
201375
201376
201377
201378
201379
201380
201381
201382
201383
201384
201385
201386
201387
201388
201389
201390
201391
201392
201393
201394
201395
201396
201397
201398
201399
201400
201401
201402
201403
201404
201405
201406
201407
201408
201409
201410
201411
201412
201413
201414
201415
201416
201417
201418
201419
201420
201421
201422
201423
201424
201425
201426
201427
201428
201429
201430
201431
201432
201433
201434
201435
201436
201437
201438
201439
201440
201441
201442
201443
201444
201445
201446
201447
201448
201449
201450
201451
201452
201453
201454
201455
201456
201457
201458
201459
201460
201461
201462
201463
201464
201465
201466
201467
201468
201469
201470
201471
201472
201473
201474
201475
201476
201477
201478
201479
201480
201481
201482
201483
201484
201485
201486
201487
201488
201489
201490
201491
201492
201493
201494
201495
201496
201497
201498
201499
201500
201501
201502
201503
201504
201505
201506
201507
201508
201509
201510
201511
201512
201513
201514
201515
201516
201517
201518
201519
201520
201521
201522
201523
201524
201525
201526
201527
201528
201529
201530
201531
201532
201533
201534
201535
201536
201537
201538
201539
201540
201541
201542
201543
201544
201545
201546
201547
201548
201549
201550
201551
201552
201553
201554
201555
201556
201557
201558
201559
201560
201561
201562
201563
201564
201565
201566
201567
201568
201569
201570
201571
201572
201573
201574
201575
201576
201577
201578
201579
201580
201581
201582
201583
201584
201585
201586
201587
201588
201589
201590
201591
201592
201593
201594
201595
201596
201597
201598
201599
201600
201601
201602
201603
201604
201605
201606
201607
201608
201609
201610
201611
201612
201613
201614
201615
201616
201617
201618
201619
201620
201621
201622
201623
201624
201625
201626
201627
201628
201629
201630
201631
201632
201633
201634
201635
201636
201637
201638
201639
201640
201641
201642
201643
201644
201645
201646
201647
201648
201649
201650
201651
201652
201653
201654
201655
201656
201657
201658
201659
201660
201661
201662
201663
201664
201665
201666
201667
201668
201669
201670
201671
201672
201673
201674
201675
201676
201677
201678
201679
201680
201681
201682
201683
201684
201685
201686
201687
201688
201689
201690
201691
201692
201693
201694
201695
201696
201697
201698
201699
201700
201701
201702
201703
201704
201705
201706
201707
201708
201709
2017010
2017011
2017012
2017013
2017014
2017015
2017016
2017017
2017018
2017019
20170100
20170101
20170102
20170103
20170104
20170105
20170106
20170107
20170108
20170109
20170110
20170111
20170112
20170113
20170114
20170115
20170116
20170117
20170118
20170119
201701100
201701101
201701102
201701103
201701104
201701105
201701106
201701107
201701108
201701109
201701110
201701111
201701112
201701113
201701114
201701115
201701116
201701117
201701118
201701119
2017011100
2017011101
2017011102
2017011103
2017011104
2017011105
2017011106
2017011107
2017011108
2017011109
2017011110
2017011111
2017011112
2017011113
2017011114
2017011115
2017011116
2017011117
2017011118
2017011119
20170111100
20170111101
20170111102
20170111103
20170111104
20170111105
20170111106
20170111107
20170111108
20170111109
20170111110
20170111111
20170111112
20170111113
20170111114
20170111115
20170111116
20170111117
20170111118
20170111119
201701111100
201701111101
201701111102
201701111103
201701111104
201701111105
201701111106
201701111107
201701111108
201701111109
201701111110
201701111111
201701111112
201701111113
201701111114
201701111115
201701111116
201701111117
201701111118
201701111119
2017011111100
2017011111101
2017011111102
2017011111103
2017011111104
2017011111105
2017011111106
2017011111107
2017011111108
2017011111109
2017011111110
2017011111111
2017011111112
2017011111113
2017011111114
2017011111115
2017011111116
2017011111117
2017011111118
2017011111119
20170111111100
20170111111101
20170111111102
20170111111103
20170111111104
20170111111105
20170111111106
20170111111107
20170111111108
20170111111109
20170111111110
20170111111111
20170111111112
20170111111113
20170111111114
20170111111115
20170111111116
20170111111117
20170111111118
20170111111119
201701111111100
201701111111101
201701111111102
201701111111103
201701111111104
201701111111105
201701111111106
201701111111107
201701111111108
201701111111109
201701111111110
201701111111111
201701111111112
201701111111113
201701111111114
201701111111115
201701111111116
201701111111117
201701111111118
201701111111119
2017011111111100
2017011111111101
2017011111111102
2017011111111103
2017011111111104
2017011111111105
2017011111111106
2017011111111107
2017011111111108
2017011111111109
2017011111111110
2017011111111111
2017011111111112
2017011111111113
2017011111111114
2017011111111115
2017011111111116
2017011111111117
2017011111111118
2017011111111119
20170111111111100
20170111111111101
20170111111111102
20170111111111103
20170111111111104
20170111111111105
20170111111111106
20170111111111107
20170111111111108
2017011111111110

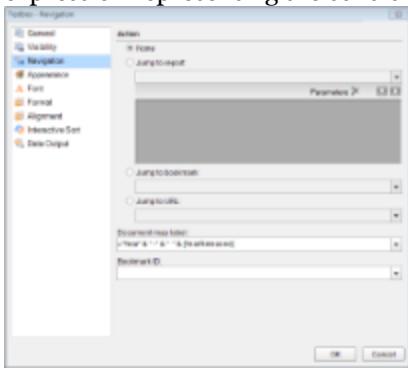


Add Items to the Document Map

In a page 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

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,

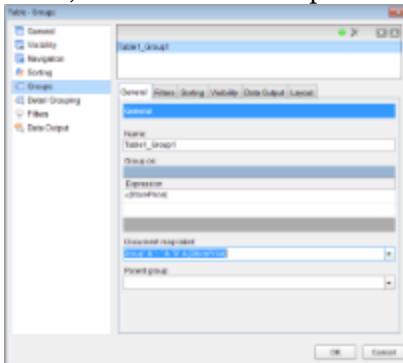
1. On the design surface, select the report control you want to add to the Document map and right-click to choose Properties from the context menu.
2. In the Properties window that appears, enter a text or an expression in the **Label** property to represent the report control in the Document map.

Go to the preview tab or the Viewer to view the document map.

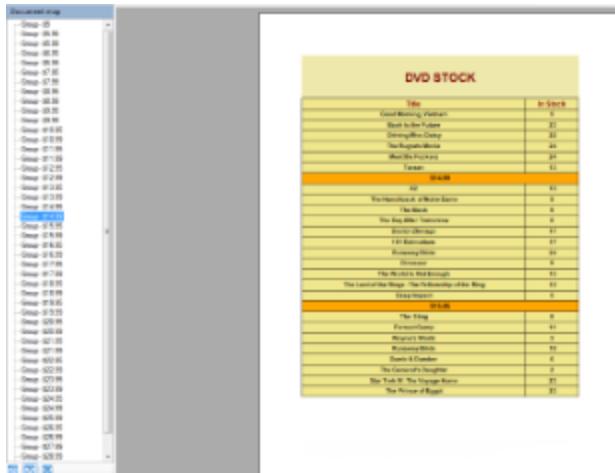


To add a group or a detail group to the Document Map

1. Open a report that contains a group. See **Grouping in a Data Region** 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.



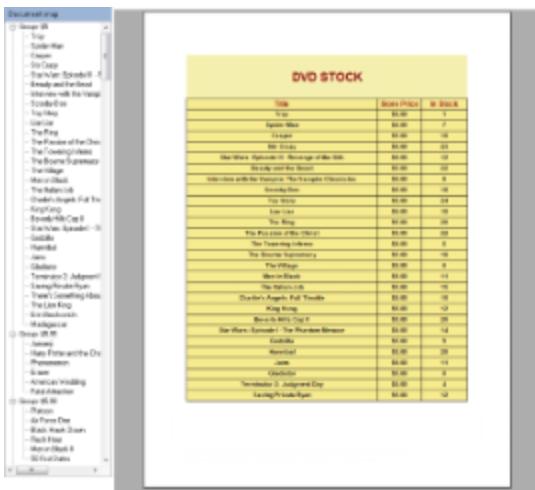
Go to the preview tab or the Viewer to view the document map.



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 **Grouping in a Data Region** 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.

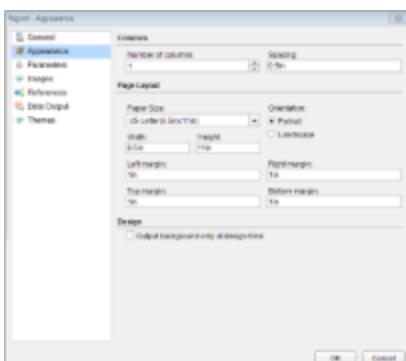
Go to the preview tab or the Viewer to view the document map.



 **Note:** In an FPL report, you can also set grouping and set the document map label on the FixedPage dialog > Groups page. See [Grouping in a FixedPage](#) to learn how to set groups on a fixed page.

Change Page Size

You can change the size of your page through the **Appearance** page of the Report Dialog.



To change the page size

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 Page Breaks in CPL

In a page layout, you can add page breaks in a CPL 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
- Matrix
- 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.

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 template and connected it to a data source. See [Adding an ActiveReport to a Project](#), [Connect to a Data Source](#) and [Add a Dataset](#) for more information.

 **Note:** This topic uses examples from the tables in the Reels database. By default, in ActiveReports Developer, the Reels.mdb file is located at [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\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.

DVD STOCK		
Country of Origin	225.00	10
Other Latin American	225.00	12
Total		247.00
The Lord of the Rings: The Two	225.00	10
Indiana Jones	225.00	10
Star Wars	225.00	12
Die Hard	225.00	10
Toy Story	225.00	10
Avatar	225.00	10
True Blood Season 2	225.00	10
Death Note	227.00	0
		Grand Total: 247

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.

SALES RECORD		
A/C No: 414-070001	Total Purchases: 321.00	
A/C No: 414-070002	Total Purchases: 321.00	
A/C No: 414-070003	Total Purchases: 321.00	
A/C No: 414-070004	Total Purchases: 319.00	
A/C No: 414-070005	Total Purchases: 319.00	
A/C No: 414-070006	Total Purchases: 319.00	
A/C No: 414-070007	Total Purchases: 319.00	
A/C No: 414-070008	Total Purchases: 319.00	
A/C No: 414-070009	Total Purchases: 319.00	
A/C No: 414-070010	Total Purchases: 319.00	
A/C No: 414-070011	Total Purchases: 319.00	
A/C No: 414-070012	Total Purchases: 321.00	
A/C No: 414-070013	Total Purchases: 321.00	
A/C No: 414-070014	Total Purchases: 321.00	
A/C No: 414-070015	Total Purchases: 321.00	
A/C No: 414-070016	Total Purchases: 321.00	
A/C No: 414-070017	Total Purchases: 321.00	
A/C No: 414-070018	Total Purchases: 321.00	
A/C No: 414-070019	Total Purchases: 321.00	
A/C No: 414-070020	Total Purchases: 321.00	

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.

SALES RECORD		
ACT#no.	044712009	Total Profit/loss
ACT#no.	044714222	Total Profit/loss
ACT#no.	044715880	Total Profit/loss
ACT#no.	044717100	Total Profit/loss
ACT#no.	044717374	Total Profit/loss
ACT#no.	044718697	Total Profit/loss
ACT#no.	044719463	Total Profit/loss
ACT#no.	044719986	Total Profit/loss
ACT#no.	044720175	Total Profit/loss
ACT#no.	044721811	Total Profit/loss
ACT#no.	044724000	Total Profit/loss
ACT#no.	044725001	Total Profit/loss
ACT#no.	044726002	Total Profit/loss
ACT#no.	044726107	Total Profit/loss
ACT#no.	044726888	Total Profit/loss

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 **BandedList** 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.

DVD STOCK		
Code	Price	In Stock
Unbreakable	15.00	5
The Bourne Identity	15.00	0
Collateral	15.00	2
Sniper	15.00	2
Die Hard 3	15.00	0
Die Hard 4	15.00	0
Die Hard With A Vengeance	15.00	0
Total:		10
Code	Price	In Stock
Gladiator	15.00	14
Master And Commander	15.00	14
Training Day	15.00	1
The Constant Gardener	15.00	7
Collateral	15.00	0
Training Day	15.00	0
The Lord Of The Rings: Return Of The King	15.00	10
The Pianist	15.00	10
Mystic River	15.00	21
Seven	15.00	0
Monsters Inc.	15.00	2
North By Northwest	15.00	0
Psycho	15.00	10
The Silence	15.00	0
Total:		100
Code	Price	In Stock
Matrix	15.00	5
The Bourne Supremacy	15.00	0
Collateral	15.00	14
The Prince Of Egypt	15.00	23
Collateral Damage	15.00	0
Monsters Inc.	15.00	0
Psycho	15.00	0
The Silence	15.00	0

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 Matrix

To add a subtotal to a matrix group

1. From the Visual Studio toolbox, drag and drop a **Matrix** data region onto the design surface.
2. From the Report Explorer, drag and drop a field (Like *Store*) onto the bottom left cell of the matrix. This is the row header, and dragging a field into it automatically adds a Row Group.
3. From the Report Explorer, drag and drop a field (Like *SaleDate*) onto the top right cell of the matrix. This is the column header, and automatically groups data in columns.

Note: In this example, we have modified the value of the SalesDate field to **=Fields!SaleDate.Value.Year** to provide yearly data.

4. From the Report Explorer, drag and drop a field (Like *SalesAmount*) onto the bottom right cell of the matrix. This is the detail data cell, and displays aggregated data for the intersection of each column and row.
5. In the Matrix data region, click the cell containing the field you placed in the last step and go to the **Properties Window**.
6. In the Properties Window, go to the **Value** property and set the expression for the detail cell to **=Sum(Fields!SalesAmount.Value)**. This displays the sum of aggregated data at the intersection of each column and row.
7. In the Matrix, right-click the column header cell and select **Subtotal**. A new column appears to the right with the text **Total**. This displays the subtotal for each row group.
8. Right-click the row header cell and select **Subtotal**. A new row appears at the bottom with the text **Total**. This displays subtotals for each column group.
9. Go to the Preview tab to view the subtotals for each year.

STORE SALES			
	Year: 2004	Year: 2005	Total Sales Rate For Year 2004+2005
Store: 1000	\$8,310.10	\$14,230.11	\$22,747.31
Store: 1001	\$4,154.40	\$9,700.83	\$13,859.32
Store: 1002	\$8,854.30	\$7,771.93	\$16,625.24
Store: 1003	\$8,726.22	\$12,080.12	\$20,806.34
Store: 1004	\$8,196.51	\$10,086.40	\$19,283.31
Store: 1005	\$8,926.81	\$11,271.47	\$19,778.32
Store: 1006	\$9,121.95	\$10,266.20	\$20,488.15
Total Yearly Sales	\$81,173.74	\$81,583.11	\$167,756.85

To add a grand total to a matrix

1. The row and columns subtotals set in the previous procedure, intersect at the rightmost cell of the Matrix 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.

STORE SALES			
	Year: 2004	Year: 2005	Total Sales Rate For Year 2004+2005
Store: 1000	\$8,310.10	\$14,230.11	\$22,747.31
Store: 1001	\$4,154.40	\$9,700.83	\$13,859.32
Store: 1002	\$8,854.30	\$7,771.93	\$16,625.24
Store: 1003	\$8,726.22	\$12,080.12	\$20,806.34
Store: 1004	\$8,196.51	\$10,086.40	\$19,283.31
Store: 1005	\$8,926.81	\$11,271.47	\$19,778.32
Store: 1006	\$9,121.95	\$10,266.20	\$20,488.15
Total Yearly Sales	\$81,173.74	\$81,583.11	\$167,756.85
			Grand Total:

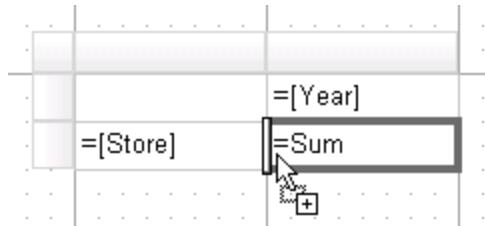
Add Static Rows and Columns to a Matrix

In a **Matrix** data region, you can add fixed rows and columns to provide space for additional detail data or labels.

To automatically add static columns and rows to a matrix

1. From the **Report Explorer**, drag and drop a field onto the bottom right (detail) cell of the Matrix data region.
2. From the Report Explorer, drag and drop an additional field onto the same detail cell to split the cell and create an additional static column or row (A grey bar indicates where the field is inserted). This creates a static cell on the selected side of the original detail cell, and adds a label using the field name.

Dropping an additional field.



New static column added.

		=[Year]	
		Month	
=[Store]	=Sum([Month])	=Sum	

Along with the additional cell, a static label cell is added automatically. The label is added at the position indicated in the following table, depending on the side of the original detail cell on which the static column or row is placed.

Position Relative to Detail Cell	Position of Label Cell	Description
Above	Left	Static row appears above the detail. An empty label cell is added so that you can label the details.
Below	Left	Static row appears below the detail. An empty label cell is added so that you can label the details.
Left	Above	Static column appears to the left of detail. An empty label cell is added so that you can label the details.
Right	Above	Static column appears to the right of detail. An empty label cell is added so that you can label the details.

To manually add static columns and rows to a matrix

1. On the design surface, right-click the bottom right (detail) cell of the Matrix data region and select **Add Column** or **Add Row**.
2. Cells are added to the Matrix at the following location as indicated in the table below. You can add text or expressions to the cells as needed.

Selection	Position Relative to Detail	Position of Label	Description
Add Row	Below	Left	Static row appears below the detail. An empty label cell is added so that you can label the details.
Add Column	Right	Above	Static row appears to the right of detail. An empty label cell is added so that you can label the details.

Cell Merging In Matrix

In the Matrix data region, cells with same values are merged automatically to delete duplicate values. The following image illustrates the result of merged cells in a Matrix where similar data exists.

Store Name	City	Name	EmployeeID
Store #1010	Beverly	Yolanda Judy	1011
		Marilyn Newell	1015
		T. E.	1018
Store #1011	Orlando	Rosa Espinoza	1048
		Laura Stevens	1050
		Marilyn Fogg	1053
Store #1012	Inglewood	Grace Krueger	1059
		Bethany	1061
		Claudia Gallegos	1069
Store #1013	Lodi	Sandra	1068
		Barnett	1070
		Angelica Adcox	1080
Store #1014	Newton	Danielle	1082
		Jeanne Marie	1085
		Yvonne	1089
Store #1015	Oak Bay	Marilyn	1090
		Arlene Herrera	1091
		Rachel Grinde	1102
Store #1016	Vi. Linn	Jillian	1113
		Marge Kalan	1121
		Ella Gandy	1125
		Vicki Gandy	1126

To allow cell merging in Matrix

The following steps use the EmployeeInfo table from the Reels database. By default, in ActiveReports Developer, the Reels.mdb file is located at [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data\Reels.mdb.

1. From the Visual Studio toolbox, add the Matrix data region onto the report design surface.
2. In the bottom left cell, set a field like **StoreName** (Fields!StoreName.Value). This automatically creates a row grouping on StoreName.
3. Right-click the Matrix to select Properties, and in the Properties Window that appears, at the bottom select **Property dialog**.
4. In the Matrix dialog that appears, go the Row Groups page and click the **Add (+)** button at the top of the group list to add another group.
5. In the expression field of this group, set an expression like Fields!City.Value. This creates an additional row grouping on City.
6. Repeat step 4 and 5 above to add two more row groups on **Name** (Fields!Name.Value) and **EmployeeID** (Fields!.EmployeeID.Value).

 **Note:** Make sure that the **Value** property of each TextBox representing the row group in the Matrix is set to the corresponding field value.

7. Click **OK** to close the dialog.
8. From the Visual Studio toolbox, place a Container control inside the merged cell of the first row in the Matrix data region to create headers for respective columns.

 **Note:** Use textbox controls inside the Container control to create a label for each column.

9. Go to the preview tab to view the result.

Cell with merged data automatically expands to accommodate large strings.

In case the data flows to the next page at a point where the merged cell is located, the merged cell value repeats on the new page along with the corresponding values of the matrix.

Page 1

Store Name	City	Name	EmployeeID
Store #1010	Beverly	Yolanda Judy	1011
		Wendie Reed	1015
		T. E.	1018
Store #1011	Orlando	Rosa Espinoza	1048
		Laura Stevens	1050
		Marilyn Fogg	1053
Store #1012	Inglewood	Grace Krueger	1059
		Bethany	1061
		Claudia Gallegos	1069
Store #1013	Lodi	Sandra	1068
		Barnett	1070
		Angelica Adcox	1080
Store #1014	Newton	Danielle	1082
		Jeanne Marie	1085
		Yvonne	1089
Store #1015	Oak Bay	Marilyn	1090
		Arlene Herrera	1091
		Rachel Grinde	1102
Store #1016	Vi. Linn	Jillian	1113
		Marge Kalan	1121
		Ella Gandy	1125
		Vicki Gandy	1126

Page 2

Store Name	City	Name	EmployeeID
Store #1010	Lodi	Angelica Adcox	1080
		Sandra	1085
		Jeanne Marie	1089
Store #1011	Newton	Yvonne	1090
		Marilyn	1091
		Arlene Herrera	1102
Store #1012	Oak Bay	Rachel Grinde	1113
		Jillian	1114
		Marge Kalan	1121
Store #1013	Vi. Linn	Ella Gandy	1125
		Vicki Gandy	1126

Section Report How To

Learn to perform common tasks in Section reports with quick how-to topics.

This section contains information about how to

Work with Data in Section Reports

Learn how to work with Section reports and perform various reporting tasks specific to this type of report.

Work with Report Controls

Learn how to work with fields, display page numbers, add charts, and many other tasks with Section report controls.

Create Common Section Reports

Learn what the tools and UI items on the report designer can help you to accomplish.

Inherit a Report Template

Learn how to inherit a report template in other reports to share common features with multiple reports.

Change Ruler Measurements

Learn how to change ruler measurements at design time and runtime.

Print Multiple Copies, Duplex and Landscape

Learn how to set duplex and landscape printing from the Report Settings dialog and also set printing for multiple copies.

Conditionally Show or Hide Details

Learn how to conditionally show or hide details in a section layout report.

Add Parameters in a Section Report

Learn how to use parameters to filter data in the report.

Add and Save Annotations

Learn how you can save a report with annotations to RDF and add annotations in a report at runtime.

Add Bookmarks

Learn how to set bookmarks.

Add Hyperlinks

Learn how to add hyperlinks in a section layout report.

Use External Style Sheets

Learn how to use external style sheets in a section layout report.

Insert or Add Pages

Learn how to insert or add pages in section layout report.

Embed Subreports

Learn how to create add a subreport to a child report in the project.

Add Code to Layouts Using Script

Learn how to add code to the layouts using script.

Export a Section Report

Learn how to export a section layout report.

Save and Load RDF Report Files

Learn how to save and load a .rdf report file.

Save and Load RPX Report Files

Learn how to save and load a .rpx report file.

Work with Data in Section Reports

See step-by-step instructions for performing common tasks using ActiveReports.

This section contains information about how to:

Bind Reports to a Data Source

Learn how to bind reports to various data sources.

Add Grouping in Section Reports

Learn how to use the GroupHeader section to group data in a section report.

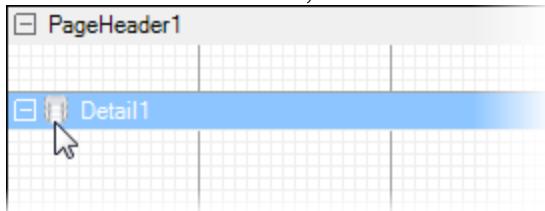
Modify Data Sources at Run Time

Learn to use code to modify a report's data source.

Bind Reports to a Data Source

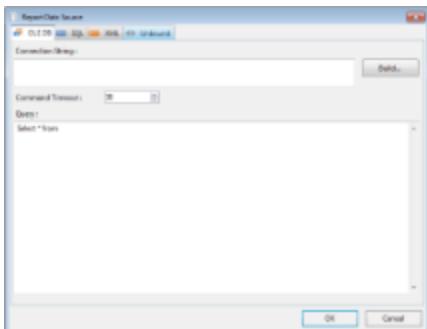
At design time, you can connect a section report to a data source through the **Report Data Source** dialog. You can access the Report Data Source dialog by doing one of the following:

- 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 7 Section Report template in a Visual Studio project. See [Adding an ActiveReport to a Project](#) for 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, 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 a SQL query to select the data that you want use from the connected database. For example, `Select * From CUSTOMERS`
7. 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, next to Connection String, click the Build button.
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`
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 located at C:\Users\YourUserName\Documents\ComponentOne Samples\ActiveReports Developer 7\Data\customer.xml.
3. In the **Recordset Pattern** field, enter a valid XPath expression like the following. `//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 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=C:\Users\YourUserName\Documents\ComponentOne
Samples\ActiveReports Developer 7\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="  
    + @"C:\Users\YourUserName\Documents\ComponentOne Samples\ActiveReports  
Developer 7\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

```
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 runtime, 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 runtime, 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.

Add Grouping in Section Reports

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 [Adding an ActiveReport to a Project](#) for further information.

To set grouping in a Section Report

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.
3. Drag and drop the grouping field onto the GroupHeader section to see the grouping field while previewing the report.

Note: You can also set a field expression in the DataField property. For example, =Country + City.

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`

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
```

Visual Basic.NET code. Paste inside the report class.

```
Private Function getDatabasePath() As String
End Function
```

This creates a function for getDatabasePath.

Visual Basic.NET code. Paste inside the getDatabasePath function.

```
Dim regKey As RegistryKey
regKey = Registry.LocalMachine
```

```
regKey = regKey.CreateSubKey("SOFTWARE\\ComponentOne\\ActiveReports Developer\\v7")
getDatabasePath = CType(regKey.GetValue(""), String)
```

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;
```

C# code. Paste inside the report class and hit Enter.

```
private string getDatabasePath()
```

This creates a function for getDatabasePath.

C# code. Paste BELOW the getDatabasePath function.

```
{
RegistryKey regKey = Registry.LocalMachine;
regKey = regKey.CreateSubKey("SOFTWARE\\ComponentOne\\ActiveReports Developer\\v7");
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();
```

Work with Report Controls

See step-by-step instructions on using Section report controls.

This section contains information about how to

Add Field Expressions

Learn how to add field expressions to a text box data field.

Display Page Numbers and Report Dates

Learn how to quickly add page numbering or report dates to Section reports using the ReportInfo control.

Load a File into a RichTextBox Control

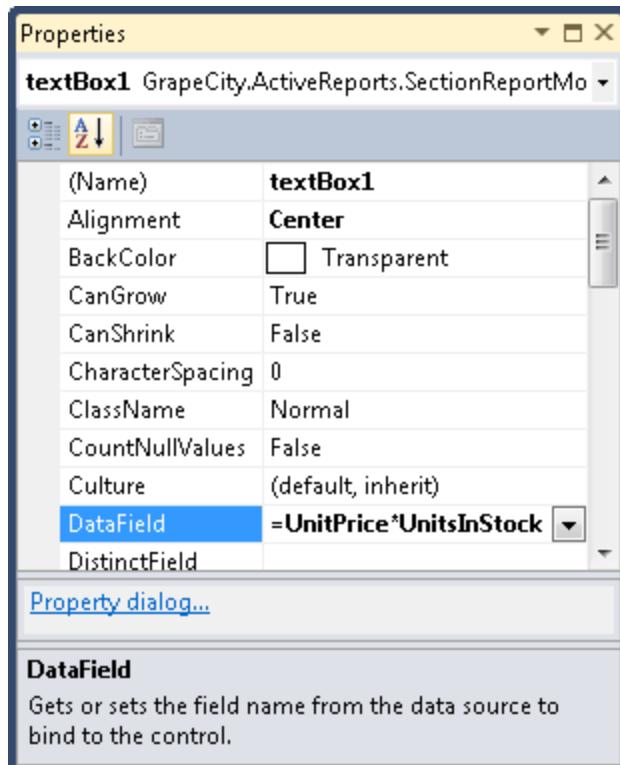
Learn how to save and load an HTML or RTF file in a RichTextBox.

Use Custom Controls on Reports

Learn how to access a third party custom control in ActiveReports.

Add Field Expressions

In a section report, expressions can be used in the **DataField ('DataField Property' in the on-line documentation)** 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 (=).



Using Field Expressions

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 Developer 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

Page {PageNumber} of {PageCount} on {RunDateTime}

Description

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 **Adding an ActiveReport to a Project** for more information.

To display page numbers and report dates on a report

1. From the ActiveReports 7 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 7 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 [Adding an ActiveReport to a Project](#) 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 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 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 *.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 DetailFormat 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

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 Developer 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 [Adding an ActiveReport to a Project](#) 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. By default, in ActiveReports, the Nwind.mdb file is located at [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\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();
```

Create Common Section Reports

See step-by-step instructions for creating commonly used reports with Section Layout.

This section contains information about how to:

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

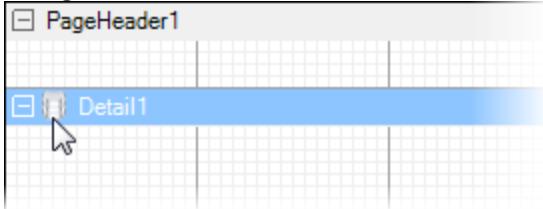
Learn to alternate background colors on the detail section.

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 sample NWind.mdb data file is located in: [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\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 **Adding an ActiveReport to a Project** and **Bind Reports to a Data Source** for further information.

Note: These steps use the Products table from the NWind database. The sample NWind.mdb database file is located in the [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data folder.

To calculate and display subtotals in a report

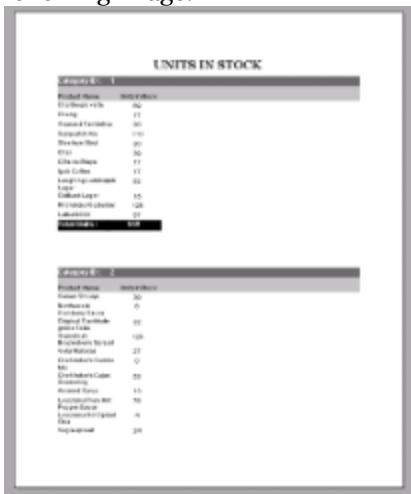
1. Right-click the design surface and select **Insert**, then **Group Header/Footer** to add group header and group footer sections to the layout.
2. With the GroupHeader section selected in the Properties Window, set its **DataField** property to *CategoryID*. This groups the data on the report according to the set field.
3. From the **Report Explorer**, drag and drop the following fields onto the corresponding sections of the report.
 - Fields**
 - **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

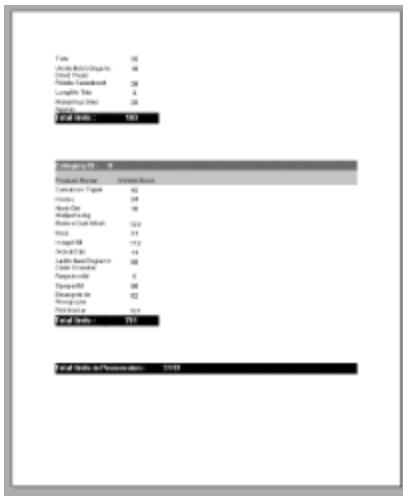
5. Go to the Preview tab to view the report and see the **Sub Total** appear below each group of data similar to the following image.



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;
}
```

- 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:



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

- In a Visual Studio project, add a Section Report (code-based) template and name it **rptLetterhead**. See **Adding an ActiveReport to a Project** for more information.
- 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.

Controls for **rptLetterhead**

Control	Section	Location	Size	Miscellaneous
Picture	PageHeader	0, 0 in	3, 0.65 in	Image = (click ellipsis and navigate to the location of your image file) 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 = http://www.grapecity.com HyperLink = http://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.



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.

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.

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) template 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.

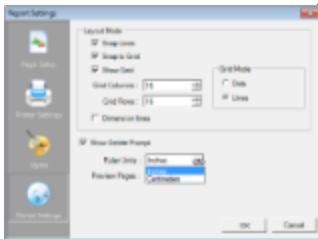
Change Ruler Measurements

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);
```

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 runtime.

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**.

Duplex Printing

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;
```

Page Orientation

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();
```

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 runtime.

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 **Adding an ActiveReport to a Project** and **Bind Reports to a Data Source** for further information.

 **Note:** These steps use the Products table from the NWind database. By default, in ActiveReports Developer, the NWind.mdb file is located in the [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\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

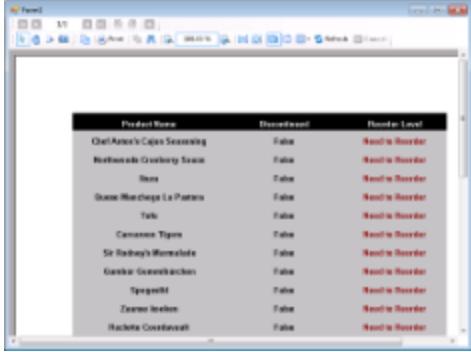
```
If Me.txtReorderLevel1.Value = 0 And Me.txtDiscontinued1.Value = False Then
    Me.Detail1.Visible = True
    Me.txtReorderLevel1.Text = "Need to Reorder"
    Me.txtReorderLevel1.ForeColor = System.Drawing.Color.DarkRed
Else
    Me.Detail1.Visible = False
End If
```

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 **Using the Viewer** for further details.
5. Press F5 to debug and see a report with discontinued products hidden from view.

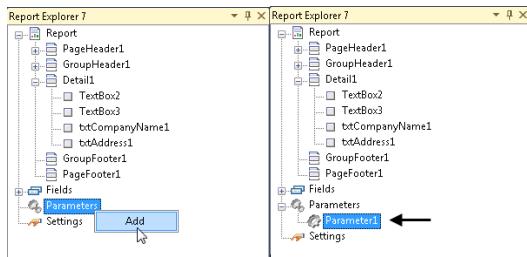


Add Parameters in a Section Report

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.



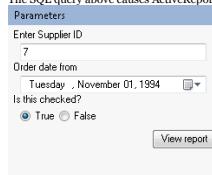
2. Select the added parameter to open the Properties Window and set values in the following properties:
 - **Name:** This is the unique name of the parameter which appears as Parameter1 by default. It corresponds to the Key property in parameters entered via code.
 - **Default Value:** Sets/returns the value displayed when the user is prompted to enter a value at runtime.
 - **Prompt:** Sets/returns a string displayed when a user is prompted for the value at runtime.
 - **PromptUser:** Boolean value indicates whether to prompt the user for a value or not. This is set True to use parameters at runtime.
 - **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 Developer displays an Enter Report Parameters dialog where the user can enter the values to fetch from the database.

1. In the detail section hand, 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 runtime.
- ```
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%>
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)
```

### 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);
```

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.

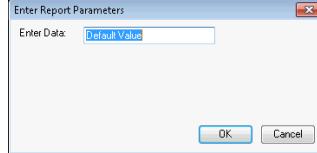
```
'Set textbox text equal to the value of the parameter.
Me.txtParam1.Text = Me.Parameters("myParam1").Value
```

### 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 runtime 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.



## Viewing 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 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 runtime. 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 [Adding an ActiveReport to a Project](#) 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");
```

## 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.Detail11.AddBookmark(CType(Me.ParentReport.Sections("Detail11").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.detail11.AddBookmark(((TextBox)
(this.ParentReport.Sections["Detail11"].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 **Viewing 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;
}

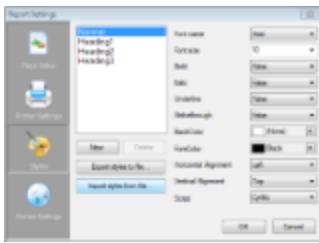
```

---

## 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 runtime 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 runtime**

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 **Adding an ActiveReport to a Project** 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;
```

## 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 7 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 [Adding an ActiveReport to a Project](#) 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

Using the **AddScriptReference** method, you can gain access to .NET or other 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();
}
```

---

### 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".

## Export a Section Report

In a section report, in order to export your reports to the various formats that ActiveReports Developer supports, you must first either add reference to the assemblies listed below or add the required export controls to your Visual Studio toolbox. For more information on how to add controls, see the **Adding ActiveReports Controls** topic.

Here are the export formats that ActiveReports Developer supports along with the necessary assembly reference:

- **HTML** For displaying on Web browsers or e-mail. You can access the HTML Export filter in code by adding the reference to *GrapeCity.ActiveReports.Export.Html.v7.dll* in your project.
- **PDF** For preserving formatting on different computers. You can access the PDF Export filter in code by adding the reference to *GrapeCity.ActiveReports.Export.Pdf.v7.dll* in your project.
- **RTF** For preserving some formatting, but allowing reports to be opened with Word or WordPad. You can access the RTF Export filter in code by adding the reference to *GrapeCity.ActiveReports.Export.Word.v7.dll* in your project.
- **Text** For transmitting raw data, with little or no formatting. You can access the Text Export filter in code by adding the reference to *GrapeCity.ActiveReports.Export.Xml.v7.dll* in your project.
- **TIFF** For transmitting faxes. You can access the Image Export filter in code by adding the reference to *GrapeCity.ActiveReports.Export.Image.v7.dll* in your project.
- **Excel** For displaying as spreadsheets. You can access the Excel Export filter in code by adding the reference to *GrapeCity.ActiveReports.Export.Excel.v7.dll* in your project.

 **Note:** HTML Export requires the .NET Framework full profile version. To ensure you are using the full profile version, go to the Visual Studio **Project** menu > **Properties** > **Compile** tab > **Advanced Compile Options** (for Visual Basic projects) or to the **Project** menu > **Properties** > **Application** tab (for C# projects) and under **Target framework** select a full profile version.

Use the following steps to export reports through export filters.

1. Place the **report.rpx** report inside your project's **Bin>Debug** folder.
2. In the Solution Explorer, right-click the References node and select **Add Reference**.
3. In the Add Reference dialog that appears, select the following references and click **OK** to add them to your project.  
*GrapeCity.ActiveReports.Export.Excel.v7*  
*GrapeCity.ActiveReports.Export.Html.v7*  
*GrapeCity.ActiveReports.Export.Image.v7*

```
GrapeCity.ActiveReports.Export.Pdf.v7
GrapeCity.ActiveReports.Export.Word.v7
GrapeCity.ActiveReports.Export.Xml.v7
```

 **Note:** The relevant export assemblies are added automatically if you add the export control to the Form. In case you have already done so, you may skip steps 2 and 3 above.

4. Double-click the Windows Form to create a Form\_Load event.
5. Add the following code to the event to export the report in various formats.

#### To write the code in Visual Basic.NET

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

```
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()
' Export the report in HTML format.
Dim htmlExport1 As New GrapeCity.ActiveReports.Export.Html.Section.HtmlExport()
htmlExport1.Export(rpt.Document, Application.StartupPath + "\\HTMLExpt.html")
' Export the report in PDF format.
Dim pdfExport1 As New GrapeCity.ActiveReports.Export.Pdf.Section.PdfExport()
pdfExport1.Export(rpt.Document, Application.StartupPath + "\\PDFExpt.pdf")
' Export the report in RTF format.
Dim rtfExport1 As New GrapeCity.ActiveReports.Export.Word.Section.RtfExport()
rtfExport1.Export(rpt.Document, Application.StartupPath + "\\RTFExpt.rtf")
' Export the report in text format.
Dim textExport1 As New GrapeCity.ActiveReports.Export.Xml.Section.TextExport()
textExport1.Export(rpt.Document, Application.StartupPath + "\\TextExpt.txt")
' Export the report in TIFF format.
Dim tiffExport1 As New
GrapeCity.ActiveReports.Export.Image.Tiff.Section.TiffExport()
tiffExport1.Export(rpt.Document, Application.StartupPath + "\\TIFFExpt.tiff")
' Export the report in Excel format.
Dim xlsExport1 As New GrapeCity.ActiveReports.Export.Excel.Section.XlsExport()
xlsExport1.FileFormat =
GrapeCity.ActiveReports.Export.Excel.Section.FileFormat.Xlsx
' Set the file format of the exported excel file to Xlsx to support Microsoft
Excel 2007 and newer versions.
xlsExport1.Export(rpt.Document, Application.StartupPath + "\\XLSExpt.xlsx")
```

---

#### To write the code in C#

#### C# code. Paste INSIDE the Form Load event.

```
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();
// Export the report in HTML format.
GrapeCity.ActiveReports.Export.Html.Section.HtmlExport htmlExport1 = new
GrapeCity.ActiveReports.Export.Html.Section.HtmlExport();
```

```
htmlExport1.Export(rpt.Document, 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(rpt.Document, 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(rpt.Document, 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(rpt.Document, 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(rpt.Document, Application.StartupPath + "\\TIFFExpt.tiff");
// Export the report in XLSX format.
GrapeCity.ActiveReports.Export.Excel.Section.XlsExport xlsExport1 = new
GrapeCity.ActiveReports.Export.Excel.Section.XlsExport();
xlsExport1.FileFormat =
GrapeCity.ActiveReports.Export.Excel.Section.FileFormat.Xlsx;
// Set the file format of the exported excel file to Xlsx to support Microsoft
Excel 2007 and newer versions.
xlsExport1.Export(rpt.Document, Application.StartupPath + "\\XLSExpt.xlsx");
```

- 
6. Press **F5** to run the application. The exported files are saved in the bin\debug folder.

## Save and Load RDF Report Files

ActiveReports Developer 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.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.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. The FlashViewer viewer type of the WebViewer (Professional Edition) may be able to display RDF files made with previous versions, but this is not guaranteed for every RDF.

## 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 Developer 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 runtime

Use the **SaveLayout** method to save your report layout at runtime.

 **Note:** When you save a report layout, ActiveReports Developer 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 runtime

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();
```

---

## Customize, 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. ComponentOne 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.

### In this section

#### ***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.

#### ***Customize the Viewer Control***

Learn how to customize the viewer control in a report.

#### ***Localize the Viewer Control***

Learn how to localize strings and images for the ActiveReports Viewer control.

#### ***Deploy Windows Applications***

Learn to deploy ActiveReports Windows applications.

#### ***Deploy Web Applications***

Learn to deploy ActiveReports Windows applications.

#### ***Localize the End User Report Designer***

Learn to localize the strings and images in the End User Report Designer.

**Customize the FlashViewer Toolbar**

Learn to deploy ActiveReports Windows applications.

**Localize the Flash Viewer**

Learn to deploy ActiveReports Windows applications.

**Configure HTTPHandlers in IIS 6.x**

Learn to deploy ActiveReports Windows applications.

**Configure HTTPHandlers in IIS 7.x**

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 ActiveReports Resources

All of the localization files are located in *C:\Program Files (x86)\ComponentOne\ActiveReports Developer 7\Localization*.

### Specify the culture you want to use in the batch files.

1. Start Notepad or another text editor as an Administrator.
2. Open \*.bat file for each resource you want to localize and change the **Culture** value to the **culture** you want to use.
3. Ensure that the path in **ProgamFilesAssemblyDir** is correct for your installation, but do not alter any of the

- other properties.
- 4. Save and close each file.

## Localize strings (and images) in the resource files.

1. Launch your zip program as an Administrator and open the \*.zip file for each resource you want to localize.
2. Extract all of the files to  
*C:\Program Files (x86)\ComponentOne\ActiveReports Developer 7\Localization*.  
A resource subfolder with the same name as the zip file is created.
3. In the new folder, open each subfolder and change the strings in each of the \*.resx files.



**Tip:** Strings are located between <value> and </value> tags in the resource files.

4. 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 files as an Administrator.

1. From the Start menu, type **cmd** in the text box, and press CTRL + SHIFT + ENTER to open a command prompt as an Administrator.
2. To change directories, type:  
**cd C:\Program Files (x86)\ComponentOne\ActiveReports Developer 7\Localization**  
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 you set inside the SatelliteAssembly folder.
  - A localized GrapeCity.ActiveReports.AssemblyName.v7.resources.dll file inside the language subfolder.
4. 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.v7.resources.dll file to [ComponentOne](#) 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" 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.CurrentCulture = 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.CurrentCulture = new
System.Globalization.CultureInfo("ja");
```

## Customize the Viewer Control

ActiveReports Developer includes a Viewer control for Windows Forms that lets you show report output in a custom preview form. You can modify the viewer toolbar and set custom commands. For example, in the following sample code we remove the Print button and add a new Print button that calls a custom dialog.

## 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 7 tab, drag the **Viewer** control onto the form. If you do not have the Viewer in your toolbox, see **Adding ActiveReports Controls**.
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 7 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.

## To add a custom print button to the viewer control

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.MainBar.Items.RemoveAt(2)
'Remove the extra separator.
Viewer1.Toolbar.MainBar.Items.RemoveAt(1)
'Add a new button to the end of the main bar with the caption "Print."
Dim tsbPrint As New ToolStripButton("Print")
Viewer1.Toolbar.MainBar.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.MainBar.Items.RemoveAt(2);
//Remove the extra separator.
viewer1.Toolbar.MainBar.Items.RemoveAt(1);
//Add a new button to the end of the main bar with the caption "Print."
```

```
ToolStripButton tsbPrint = new ToolStripButton("Print");
viewer1.Toolbar.MainBar.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 see the custom print button on the viewer.

You can also remove any of the other buttons from the three toolbars: MainBar, MouseModeBar, and NavigationBar. For this example, we only added a button with text, but you can also add an icon or an icon plus text. You can see custom icons, custom menus, and more in the included Custom Preview sample that is located in  
...\\Documents\\ComponentOne Samples\\ActiveReports Developer 7\\Section Reports\\C#\\CustomPreview.

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

## 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)\\ComponentOne\\ActiveReports Developer 7\\Localization*.

## Specify the culture you want to use in the batch file.

1. Start Notepad or another text editor as an Administrator.
2. Open the WinViewer.bat file and change the **Culture** value to the **culture** you want to use.
3. Ensure that the path in **ProgamFilesAssemblyDir** is correct for your installation, but do not alter any of the other properties.
4. Save and close the file.

## Localize strings (and images) in the resource files.

1. Launch your zip program as an Administrator and open the WinViewer.zip file.
2. Extract all of the files to  
*C:\Program Files (x86)\ComponentOne\ActiveReports Developer 7\Localization*.  
A WinViewer subfolder is created.
3. In the new WinViewer folder, open each subfolder and change the strings in each of the \*.resx files.



**Tip:** Strings are located between <value> and </value> tags in the resource files.

4. 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 as an Administrator.

1. From the Start menu, type **cmd** in the text box, and press CTRL + SHIFT + ENTER to open a command prompt as an Administrator.
2. To change directories, type:  
**cd C:\Program Files (x86)\ComponentOne\ActiveReports Developer 7\Localization**  
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 you set inside the SatelliteAssembly folder.
  - A localized GrapeCity.ActiveReports.Viewer.Win.v7.resources.dll file inside the language subfolder.
4. 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.v7.resources.dll file to [ComponentOne](#) 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.CurrentCulture = 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.CurrentCulture = 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:<br>Excel, Html, Image/Tiff, Pdf, Word/Rtf, Text/Xml | GrapeCity.ActiveReports.Export.*.v7.dll<br>(replace * with Excel, Html, Image, Pdf, Word, or Xml) |
| Chart report control                                        | GrapeCity.ActiveReports.Chart.v7.dll                                                              |
| Calendar report control                                     | GrapeCity.ActiveReports.Calendar.v7.dll                                                           |
| Sparkline or Bullet report control                          | GrapeCity.ActiveReports.Dashboard.v7.dll                                                          |
| Viewer control (or printing without the Viewer control)     | GrapeCity.ActiveReports.Viewer.Win.v7.dll                                                         |
| Designer, Toolbox, or ReportExplorer control                | GrapeCity.ActiveReports.Design.Win.v7.dll                                                         |

## XCopy Deployment

1. Open your project in Visual Studio, and from the Build menu, select **Build Solution**.
2. In Windows Explorer, navigate to the project's bin directory, and copy the files from the **Release** folder into a zip file.
3. 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 or Interop64.dll for 64-bit machines), 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 [powersupport@grapecity.com](mailto:powersupport@grapecity.com).

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 6.x** or **Configure HTTPHandlers in IIS 7.x**.

## Before you begin

To deploy ActiveReports Web projects, you must have access to the Microsoft .NET Framework version 3.5 SP1 or higher and the coordinating version of ASP.NET. You must also have access to Internet Information Services version 5.1 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 Assembly                                                                                 |
|-------------------------------------------------------------|---------------------------------------------------------------------------------------------------|
| Export:<br>Excel, Html, Image/Tiff, Pdf, Word/Rtf, Text/Xml | GrapeCity.ActiveReports.Export.*.v7.dll<br>(replace * with Excel, Html, Image, Pdf, Word, or Xml) |
| Chart report control                                        | GrapeCity.ActiveReports.Chart.v7.dll                                                              |
| Calendar report control                                     | GrapeCity.ActiveReports.Calendar.v7.dll                                                           |
| Sparkline or Bullet report control                          | GrapeCity.ActiveReports.Dashboard.v7.dll                                                          |
| WebViewer or HttpHandlers (Pro Edition only)                | GrapeCity.ActiveReports.Web.v7.dll                                                                |

### To copy referenced DLLs to your project

1. In the Visual Studio Solution Explorer, if the References node is not showing, click the **Show All Files** button.

2. Expand the **References** node, and select one of the ActiveReports references.
3. In the Properties window, change the **CopyLocal** property to **True**. The corresponding DLL is stored in the Bin folder of your project.
4. Set the **CopyLocal** property to **True** for each ActiveReports reference used in your project.

## 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 3.5 SP1 or higher
- Internet Information Services (IIS) version 5.1 or 6.0
- ASP.NET version 3.5 or higher (must be the same version as the Framework)

## To copy your project to the server

1. Copy the entire directory containing your project to the server.
2. If your project is in a virtual directory on your local machine (i.e. C:\Inetpub\wwwroot\YourProject), you must set up a virtual directory in IIS on the server as well.

## 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 XP or 2000 machines, the **ASPNET** user ID needs **Write** access to that folder.
- Windows 2003 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.

## 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.

## To localize the Designer control

All of the localization files are located in *C:\Program Files (x86)\ComponentOne\ActiveReports Developer 7\Localization*.

### Specify the culture you want to use in the batch file.

1. Start Notepad or another text editor as an Administrator.
2. Open the **ARDesigner.bat** file and change the **Culture** value to the **culture** you want to use.
3. Ensure that the path in **ProgamFilesAssemblyDir** is correct for your installation, but do not alter any of the other properties.
4. Save and close the file.

### Localize strings (and images) in the resource files.

1. Launch your zip program as an Administrator and open the ARDesigner.zip file.
2. Extract all of the files to  
*C:\Program Files (x86)\ComponentOne\ActiveReports Developer 7\Localization*.  
An ARDesigner subfolder is created.
3. In the new folder, open each subfolder and change the strings in each of the \*.resx files.



**Tip:** Strings are located between <value> and </value> tags in the resource files.

- 
4. 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 as an Administrator.

1. From the Start menu, type **cmd** in the text box, and press CTRL + SHIFT + ENTER to open a command prompt as an Administrator.
2. To change directories, type:  
**cd C:\Program Files (x86)\ComponentOne\ActiveReports Developer 7\Localization**  
and press Enter.
3. Type **ARDesigner.bat** and press Enter to run the file. The NameCompleter.exe application runs, and creates the following.
  - A SatelliteAssembly folder inside the ARDesigner subfolder.
  - A language subfolder with the name of the culture you set inside the SatelliteAssembly folder.
  - A localized GrapeCity.ActiveReports.Design.Win.v7.resources.dll file inside the language subfolder.
4. 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.Design.Win.v7.resources.dll file to [ComponentOne](#) 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" in the example code with the culture you specified in the ARDesigner.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.CurrentCulture = 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.CurrentCulture = new
System.Globalization.CultureInfo("ja");
```

## Customize the FlashViewer Toolbar

When you use the WebViewer that is licensed with the Professional Edition, one of the ViewerTypes that you can select is FlashViewer. The FlashViewer toolbar is very similar to the Viewer control's toolbar. You can show or hide it, reorder buttons, remove buttons, add custom buttons, or create a custom toolbar. Use the **Web.Controls ('GrapeCity.ActiveReports.Web.Controls Namespace' in the on-line documentation)** namespace to create custom buttons or a custom toolbar that you can specify in the WebViewer's FlashViewerToolbar property.

The following buttons are provided by default.

|         |             |                |
|---------|-------------|----------------|
| Heading | Print       | Page Range     |
| Search  | Zoom-out    | Zoom           |
| Zoom-in | Single Page | Multiple pages |

[Continuous Page](#)[Previous Page](#)[Next Page](#)[Page Number](#)[Backwards](#)[Forward](#)

Drop down the sections below for code samples.

 **Note:** This code is ignored if the **ViewerType** property of the WebViewer control is not set to **FlashViewer**.

## To hide the toolbar

1. In the Visual Studio Solution Explorer, right-click the ASPX file that contains your WebViewer and select **View Designer**.
2. In the design view of your web form, double-click on the form below the WebViewer. This creates an event handling method for the Page Load event and takes you to the code view of the page.
3. Use the following code to hide the toolbar.

### Paste INSIDE the Page Load event

```
WebViewer1.FlashViewerToolBar.Visible = False
```

---

### Paste INSIDE the Page Load event

```
WebViewer1.FlashViewerToolBar.Visible = false;
```

---

To reduce the amount of code needed for the rest of the customizations, add a using or Imports directive at the top of the ASPX code view.

## To add a using or Imports directive

### Paste at the top of the ASPX code view

```
Imports GrapeCity.ActiveReports.Web.Controls
```

---

### Paste near the top of the ASPX code view

```
using GrapeCity.ActiveReports.Web.Controls;
```

---

## To reorder buttons in the toolbar

1. In the Visual Studio Solution Explorer, right-click the ASPX file that contains your WebViewer and select **View Designer**.
2. In the design view of your web form, double-click on the form below the WebViewer. This creates an event handling method for the Page Load event and takes you to the code view of the page.
3. Use the following code to create a new button and add it at the beginning of the toolbar.

### Paste INSIDE the Page Load event

```
'Get a default tool from the toolbar. (You can also specify the tool index.)
Dim tool As ToolBase = WebViewer1.FlashViewerToolBar.Tools("PageRangeButton")
'Remove the tool from the toolbar.
WebViewer1.FlashViewerToolBar.Tools.Remove(tool)
'Insert the tool in a different position.
WebViewer1.FlashViewerToolBar.Tools.Insert(0, tool)
```

---

### Paste INSIDE the Page Load event

```
//Get a default tool from the toolbar. (You can also specify the tool index.)
ToolBase tool = WebViewer1.FlashViewerToolBar.Tools["PageRangeButton"];
//Remove the tool from the toolbar.
WebViewer1.FlashViewerToolBar.Tools.Remove(tool);
//Insert the tool in a different position.
```

```
WebViewer1.FlashViewerToolBar.Tools.Insert(0, tool);
```

## To remove a button from the toolbar

1. In the Visual Studio Solution Explorer, right-click the ASPX file that contains your WebViewer and select **View Designer**.
2. In the design view of your web form, double-click on the form below the WebViewer. This creates an event handling method for the Page Load event and takes you to the code view of the page.
3. Use the following code to remove a button from the toolbar.

### Paste INSIDE the Page Load event

```
'Get a default tool from the toolbar by name. (You can also specify the tool index.)
Dim tool As ToolBase = WebViewer1.FlashViewerToolBar.Tools("PageRangeButton")
' Delete the tool from the toolbar.
WebViewer1.FlashViewerToolBar.Tools.Remove(tool)
```

### Paste INSIDE the Page Load event

```
//Get a default tool from the toolbar by name. (You can also specify the tool index.)
ToolBase tool = WebViewer1.FlashViewerToolBar.Tools["PageRangeButton"];
//Delete the tool from the toolbar.
WebViewer1.FlashViewerToolBar.Tools.Remove(tool);
```

## To create a custom button and add it to the toolbar

 **Tip:** The ToolsCollection class in the Web.Controls namespace has the standard System.Collections.ObjectModel.Collection methods available, so if you want to just add a button to the end of the toolbar, you can use the **Add** method instead.

1. In the Visual Studio Solution Explorer, right-click the ASPX file that contains your WebViewer and select **View Designer**.
2. In the design view of your web form, double-click on the form below the WebViewer. This creates an event handling method for the Page Load event and takes you to the code view of the page.
3. Use the following code to create a button and place it at the beginning of the toolbar.

### Paste INSIDE the Page Load event

```
Dim customButton As ToolButton = Tool.CreateButton("CustomButton")
customButton.Caption = "Contact Us"
customButton.ToolTip = "ComponentOne Website"
customButton.ClickNavigateTo = "http://www.componentone.com"
'Add a button at the specified index.
'An index value of 20 places it second to last, between the Forward and Backward buttons.
'Set the index value to 0 to place it in the first position at the left.
WebViewer1.FlashViewerToolBar.Tools.Insert(20, customButton)
```

### Paste INSIDE the Page Load event

```
ToolButton customButton = Tool.CreateButton("CustomButton");
customButton.Caption = "Contact Us";
customButton.ToolTip = "ComponentOne Website";
customButton.ClickNavigateTo = "http://www.componentone.com";
'Add a button at the specified index.
'An index value of 20 places it second to last, between the Forward and Backward buttons.
```

```
//Set the index value to 0 to place it in the first position at the left.
WebViewer1.FlashViewerToolBar.Tools.Insert(20, customButton);
```

## To create a custom toolbar and add it to the viewer

1. In the Visual Studio Solution Explorer, right-click the ASPX file that contains your WebViewer and select **View Designer..**
2. In the design view of your web form, double-click on the form below the WebViewer. This creates an event handling method for the Page Load event and takes you to the code view of the page.
3. Use the following code to create the custom toolbar and add it to viewer.

### Paste INSIDE the Page Load event

```
'Get the collection of buttons and separators used in the toolbar.
Dim collection As ToolsCollection = WebViewer1.FlashViewerToolBar.Tools
'Delete all of the buttons and separators from the toolbar.
collection.Clear()
'Add pre-defined buttons.
collection.Add(Tool.Create(ToolsCollection.ToolCommands.ZoomOutButton))
collection.Add(Tool.Create(ToolsCollection.ToolCommands.ZoomBox))
collection.Add(Tool.Create(ToolsCollection.ToolCommands.ZoomInButton))
' Add a separator.
collection.Add(Tool.CreateSeparator())
'Add a pre-defined button.
collection.Add(Tool.Create(ToolsCollection.ToolCommands.SearchButton))
'Add a separator.
collection.Add(Tool.CreateSeparator())
'Add custom buttons.
collection.Add(Tool.CreateButton("btn1"))
collection.Add(Tool.CreateButton("btn2"))
```

### Paste INSIDE the Page Load event

```
//Get the collection of buttons and separators used in the toolbar.
ToolsCollection collection = WebViewer1.FlashViewerToolBar.Tools;
//Delete all of the buttons and separators from the toolbar.
collection.Clear();
//Add pre-defined buttons.
collection.Add(Tool.Create(ToolsCollection.ToolCommands.ZoomOutButton));
collection.Add(Tool.Create(ToolsCollection.ToolCommands.ZoomBox));
collection.Add(Tool.Create(ToolsCollection.ToolCommands.ZoomInButton));
//Add a separator.
collection.Add(Tool.CreateSeparator());
//Add a pre-defined button.
collection.Add(Tool.Create(ToolsCollection.ToolCommands.SearchButton));
//Add a separator.
collection.Add(Tool.CreateSeparator());
//Add custom buttons.
collection.Add(Tool.CreateButton("btn1"));
collection.Add(Tool.CreateButton("btn2"));
```

## Localize the Flash Viewer

The FlashViewer, one of the ViewerTypes of the WebViewer control, is localized separately from other ActiveReports resources. You can redistribute the Flash localization resources separately from the application so that you need not recompile the GrapeCity.ActiveReports.Flash.v7.swf file.

You can localize all of the UI strings, error messages, fonts, and images that appear in the FlashViewer in the included resource file, and send it to ComponentOne to be compiled into the SWF resources file.

 **Note:** If you are willing to share your new localization with other customers, let support know so that the updated GrapeCity.ActiveReports.Flash.v7.Resources.swf file can be included in future builds of the product.

## Included Localizations

The default locale is en\_US, U.S. English, but the included GrapeCity.ActiveReports.Flash.v7.Resources.swf file also contains strings localized for the following languages:

- **ru\_RU** Russian
- **ja\_JP** Japanese
- **zh\_CN** Simplified Chinese

The SWF files are located in *C:\Program Files (x86)\ComponentOne\ActiveReports Developer 7\Deployment\Flash*.

### To use the Russian, Japanese, or Chinese localization

1. From *C:\Program Files (x86)\ComponentOne\ActiveReports Developer 7\Deployment\Flash*, copy **GrapeCity.ActiveReports.Flash.v7.Resources.swf** into the project folder that contains the ASPX file with the WebViewer.
2. In the Design view of the ASPX file, click the WebViewer control to select it, and in the Properties window, expand the **FlashViewerOptions** property node.
3. In the **ResourceLocale** property, drop down the list of values and select the locale that you want to use.
4. Run the project to see the localized FlashViewer.

## Custom Localizations

The localization file is located in *C:\Program Files (x86)\ComponentOne\ActiveReports Developer 7\Localization*.

### To localize strings (and images) in the resource file.

1. In *C:\Program Files (x86)\ComponentOne\ActiveReports Developer 7\Localization*, open the **FlashViewer.zip** file.
2. Using Notepad, open the **Resources.properties** file and localize the strings.
3. Save and zip the file.
4. If you want to change the toc.png image for the Table of Contents button icon, rename your localized image to toc.png and add it to the zip file.

### To send the localized file to ComponentOne to be compiled into the SWF file.

1. Attach the localized FlashViewer.zip file to the Submit New Issue form on our web site: <http://www.componentone.com/Support/default.aspx?new=true>.
2. Tell support the **culture** of the localization and ask them to compile it into the GrapeCity.ActiveReports.Flash.v7.Resources.swf.
3. Support will email you the localized SWF file.

### To use your custom localization.

1. When you receive the new ActiveReports.FlashViewer.Resources.swf file, copy it into the project folder that contains the ASPX file with the WebViewer.
2. In the Design view of the ASPX file, click the WebViewer control to select it, and in the Properties window, expand the **FlashViewerOptions** property node.
3. In the **ResourceLocale** property, drop down the list of values and select your custom locale.
4. Run the project to see the localized FlashViewer.

## Configure HTTPHandlers in IIS 6.x

HttpHandlers are included in the Professional edition of ActiveReports to allow you to quickly and easily display reports in the browser. In order to use ActiveReports HTTP handlers on the Web with ASP.NET, you must first configure the machine to use the handlers.

For information on how to configure ActiveReports handler mappings in IIS 7.x, see the section **Configure HTTPHandlers in IIS 7.x** of this Guide.

Follow these steps to configure the ActiveReports HTTP handlers in IIS 6.x 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.

### To configure ActiveReports HTTP handlers to enable report linking on your machine

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, expand the **Sites** node, right-click the Web application you want to configure, and select **Properties**.
3. On the Directory tab of the *YourWebSite* Properties dialog that appears, click the **Configuration** button.
4. In the Application Configuration dialog that appears, select the list item with **.aspx** in the Extension column and click **Edit**.

 **Note:** If your machine does not have the ASP.NET server components installed, the **.aspx** handler does not appear in the Application Mappings list.

5. In the **Executable** field, select and copy all of the text, and click **Cancel** to return to the Application Configuration dialog.
6. Click the **Add** button to add a new Application Mapping.
7. In the Add/Edit Application Extension Mapping dialog that appears, enter the information from the first row of the table below.

#### Executable

Paste the text copied from the \*.aspx script map.  
Paste the text copied from the \*.aspx script map.  
Paste the text copied from the \*.aspx script map.  
Paste the text copied from the \*.aspx script map.

#### Extension

.ar7  
.ar7Web  
.rpx  
.rdlx, .rdl

#### Check that file exists

Clear this checkbox.  
Clear this checkbox.  
Select this checkbox.  
Select this checkbox.

8. Click **OK** to close the window and add the script mapping.
9. Repeat for each script mapping in the table above.

### To enable HTTP handlers in your project

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 **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.v7, Version=7.0.5252.0, Culture=neutral,
PublicKeyToken=cc496777c49a3ff" />
 <add verb="*" path="*.rdlx, *.rdl"
type="GrapeCity.ActiveReports.Web.Handlers.RdlxHandler, GrapeCity.ActiveReports.Web.v7,
Version=7.0.5252.0, Culture=neutral, PublicKeyToken=cc496777c49a3ff" />
 <add verb="*" path="*.ar7"
type="GrapeCity.ActiveReports.Web.Handlers.ReportBinariesStreamer,
GrapeCity.ActiveReports.Web.v7, Version=7.0.5252.0, Culture=neutral,
```

```
PublicKeyToken=cc4967777c49a3ff" />
<add verb="*" path="*.ar7Web"
type="GrapeCity.ActiveReports.Web.Handlers.WebCacheAccessHandler,
GrapeCity.ActiveReports.Web.v7, Version=7.0.5252.0, Culture=neutral,
PublicKeyToken=cc4967777c49a3ff" />
</httpHandlers>
```

## Configure HTTPHandlers in IIS 7.x

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 7.x 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.

## 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
*.ar7	aspnet_isapi.dll version to match your app pool	ActiveReports 7 Script Mapping
*.ar7Web	aspnet_isapi.dll version to match your app pool	ActiveReports 7 Cache Item Script Mapping
*.rpx	aspnet_isapi.dll version to match your app pool	ActiveReport 7 RPX Script Mapping
*.rdlx, *.rdl	aspnet_isapi.dll version to match your app pool	ActiveReports 7 RDLX 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 7.x 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.v7, Version=7.0.5252.0, Culture=neutral,
 PublicKeyToken=cc496777c49a3ff" />
 <add verb="*" path="*.rdlx, *.rdl"
 type="GrapeCity.ActiveReports.Web.Handlers.RdlxHandler,
 GrapeCity.ActiveReports.Web.v7, Version=7.0.5252.0, Culture=neutral,
 PublicKeyToken=cc496777c49a3ff" />
 <add verb="*" path="*.ar7"
 type="GrapeCity.ActiveReports.Web.Handlers.ReportBinariesStreamer,
 GrapeCity.ActiveReports.Web.v7, Version=7.0.5252.0, Culture=neutral,
 PublicKeyToken=cc496777c49a3ff" />
 <add verb="*" path="*.ar7Web"
 type="GrapeCity.ActiveReports.Web.Handlers.WebCacheAccessHandler,
 GrapeCity.ActiveReports.Web.v7, Version=7.0.5252.0, Culture=neutral,
 PublicKeyToken=cc496777c49a3ff" />
</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 2.0 or 4.0 installed on your machine:

#### .Net 2.0

### Paste inside the <system.webServer> tags.

```
<handlers>
 <add name="AR7Rpx" path="*.rpx" verb="*" modules="IsapiModule"
 scriptProcessor="%windir%\Microsoft.NET\Framework\v2.0.50727\aspnet_isapi.dll"
 preCondition="classicMode,runtimeVersionv2.0,bitness32"/>
 <add name="AR7Rdlx" path="*.rdlx" verb="*" modules="IsapiModule"
 scriptProcessor="%windir%\Microsoft.NET\Framework\v2.0.50727\aspnet_isapi.dll"
 preCondition="classicMode,runtimeVersionv2.0,bitness32"/>
 <add name="AR7Rdl" path="*.rdl" verb="*" modules="IsapiModule"
 scriptProcessor="%windir%\Microsoft.NET\Framework\v2.0.50727\aspnet_isapi.dll"
 preCondition="classicMode,runtimeVersionv2.0,bitness32"/>
 <add name="AR7" path="*.ar7" verb="*" modules="IsapiModule"
 scriptProcessor="%windir%\Microsoft.NET\Framework\v2.0.50727\aspnet_isapi.dll"
 preCondition="classicMode,runtimeVersionv2.0,bitness32"/>
 <add name="AR7Web" path="*.ar7Web" verb="*" modules="IsapiModule"
 scriptProcessor="%windir%\Microsoft.NET\Framework\v2.0.50727\aspnet_isapi.dll"
 preCondition="classicMode,runtimeVersionv2.0,bitness32"/>
</handlers>
```

#### .Net 4.0

### Paste inside the <system.webServer> tags.

```
<handlers>
 <add name="AR7Rpx" path="*.rpx" verb="*" modules="IsapiModule"
scriptProcessor="%windir%\Microsoft.NET\Framework\v4.0.30319\aspnet_isapi.dll"
preCondition="classicMode,runtimeVersionv2.0,bitness32"/>
 <add name="AR7Rdlx" path="*.rdlx" verb="*" modules="IsapiModule"
scriptProcessor="%windir%\Microsoft.NET\Framework\v4.0.30319\aspnet_isapi.dll"
preCondition="classicMode,runtimeVersionv2.0,bitness32"/>
 <add name="AR7Rdl" path="*.rdl" verb="*" modules="IsapiModule"
scriptProcessor="%windir%\Microsoft.NET\Framework\v4.0.30319\aspnet_isapi.dll"
preCondition="classicMode,runtimeVersionv2.0,bitness32"/>
 <add name="AR7" path="*.ar7" verb="*" modules="IsapiModule"
scriptProcessor="%windir%\Microsoft.NET\Framework\v4.0.30319\aspnet_isapi.dll"
preCondition="classicMode,runtimeVersionv2.0,bitness32"/>
 <add name="AR7Web" path="*.ar7Web" verb="*" modules="IsapiModule"
scriptProcessor="%windir%\Microsoft.NET\Framework\v4.0.30319\aspnet_isapi.dll"
preCondition="classicMode,runtimeVersionv2.0,bitness32"/>
</handlers>
```

 **Note:** If you have a 64 bit Web application, change the **preCondition** attribute on each line to **classicMode, runtimeVersionv2.0,bitness64**, or in **ASP.NET 4**, change it to **classicMode, runtimeVersion4.0,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
*.ar7	GrapeCity.ActiveReports.Web.Handlers.ReportBinariesStreamer	ActiveReports 7 integrated handler mapping
*.ar7Web	GrapeCity.ActiveReports.Web.Handlers.WebCacheAccessHandler	ActiveReports 7 cache item integrated handler mapping
*.rpx	GrapeCity.ActiveReports.Web.Handlers.RpxHandler	ActiveReports 7 RPX integrated handler mapping
*.rdlx, *.rdl	GrapeCity.ActiveReports.Web.Handlers.RdlxHandler	ActiveReports 7 RDLX 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 7.x 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.

```
<handlers>
 <add name="*.rpx_*" path="*.rpx" verb="*"
type="GrapeCity.ActiveReports.Web.Handlers.RpxHandler, GrapeCity.ActiveReports.Web.v7,
Version=7.0.5252.0, Culture=neutral, PublicKeyToken=cc496777c49a3ff"
preCondition="integratedMode, runtimeVersionv2.0" />
 <add name="*.rdlx_*" path="*.rdlx, *.rdl" verb="*"
type="GrapeCity.ActiveReports.Web.Handlers.RdlxHandler, GrapeCity.ActiveReports.Web.v7,
Version=7.0.5252.0, Culture=neutral, PublicKeyToken=cc496777c49a3ff"
preCondition="integratedMode, runtimeVersionv2.0" />
 <add name="*.ActiveReport7_*" path="*.ar7" verb="*"
type="GrapeCity.ActiveReports.Web.Handlers.ReportBinariesStreamer,
GrapeCity.ActiveReports.Web.v7, Version=7.0.5252.0, Culture=neutral,
PublicKeyToken=cc496777c49a3ff" preCondition="integratedMode, runtimeVersionv2.0" />
 <add name="*.ArCacheItem_*" path="*.ar7Web" verb="*"
type="GrapeCity.ActiveReports.Web.Handlers.WebCacheAccessHandler,
GrapeCity.ActiveReports.Web.v7, Version=7.0.5252.0, Culture=neutral,
PublicKeyToken=cc496777c49a3ff" preCondition="integratedMode, runtimeVersionv2.0" />
</handlers>
```

 **Note:** If you have a 64 bit Web application, change the **preCondition** attribute on each line to **integratedMode, runtimeVersionv2.0,bitness64**, or in **ASP.NET 4**, change it to **integratedMode, runtimeVersion4.0,bitness64**.

## Use Fields in Reports

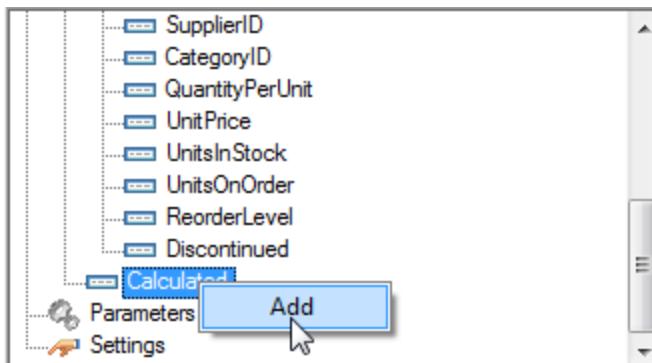
Fields provide data to display on a report page. ActiveReports has two types of fields; a bound or database field and a calculated field.

- **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.
- **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**.



- 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**.

(Name)	Inventory
DefaultValue	
FieldType	<b>None</b>
Formula	<b>=UnitsInStock - ReorderLevel</b>

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

- 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.

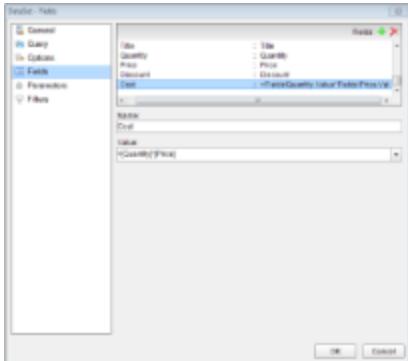
### To create a calculated field in a Page Report

In a page 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.



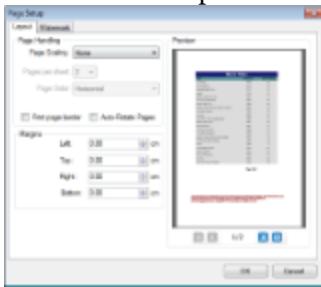
3. Under **Name**, enter the name of the field. By default it appears as Field1.
4. Under **Value**, click the dropdown arrow and select <Expression...>, to open the Expression Editor dialog.
5. In the Expression Editor dialog, create an expression you want to use as the value for the calculated field. 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.

## Use Advanced Printing 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 **Using the 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.

## Provide One-Touch Printing in the WebViewer (Pro Edition)

In the WebViewer control, when the **ViewerType** is set to FlashViewer, you can provide one-touch printing. This opens the Print dialog as soon as you run your application. This feature is available in the Professional Edition only.

The following steps assume that you have already created a Web Application in Visual Studio and added a WebViewer control on the aspx page. See **Adding ActiveReports Controls** and **Getting Started with the WebViewer** for further information.

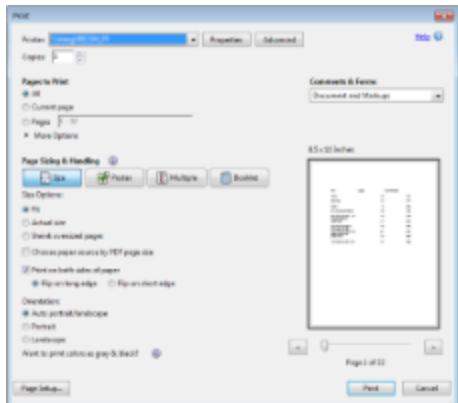
### To provide one-touch printing

1. Add an ActiveReport to the project. See **Adding an ActiveReport to a Project** for further details.
2. Go to the aspx page of your application which contains the WebViewer.
3. With the WebViewer control selected, go to the Properties Window, and make the following changes:
  - Set the **ReportName** property to the name of your report.
  - Set the **ViewerType** property to FlashViewer.
  - Expand the **FlashViewerOptions > PrintOptions** node, and set the **StartPrint** property to True.
  - If you do not want to display the report to the user, set the **Height** and **Width** properties to 0.
4. Copy the **ActiveReports.FlashViewer.swf** file into your project folder where your aspx file is located. You can get this file from the ...\\ComponentOne\\ActiveReports Developer 7\\Deployment\\Flash folder.
5. Run the project to see the **Print** dialog box and send the report to print by clicking the **Print** button.

## Provide PDF Printing in the Silverlight Viewer (Pro Edition)

You can set PDF printing in your Silverlight project to allow printing a document from Silverlight to the PDF format directly. This is a good alternative to the default Silverlight printing with its large print spool size issue.

If PDF printing is set up in the Silverlight project, you see a Print dialog appear on clicking the **Print** button in the SilverlightViewer toolbar:



When you print to PDF, the exceptions that occur at printing cannot be caught. Thus, if there is a printing exception, no Print dialog appears after you click the **Print** button.

**Caution:** If your Web browser does not support JavaScript and Adobe Reader, the Silverlight Viewer will use its default printing instead of the PDF printing. Also, PDF printing is not supported in the Silverlight Out-of-Browser applications.

## To set up PDF printing in the Silverlight Viewer project

1. Open your Silverlight project or create a new Silverlight project as described in [Using the SilverlightViewer](#).
2. In Solution Explorer, open the Web.config file.
3. Make sure that the http handlers have been added to the Web.config file. The http handlers are added to the Web config automatically when you add **ActiveReports Developer 7 Web Service** - see [Using the Silverlight Viewer](#) for details:

**XML code. Handlers appear in the XML view of the Web.config file inside the system.web section.**

```
<httpHandlers>
 <add verb="*" path="*.ar7"
 type="GrapeCity.ActiveReports.Web.Handlers.ReportBinariesStreamer,
 GrapeCity.ActiveReports.Web.v7, Version=7.0.xxxx.0, Culture=neutral,
 PublicKeyToken=cc496777c49a3ff" />
 <add verb="*" path="*.ar7Web"
 type="GrapeCity.ActiveReports.Web.Handlers.WebCacheAccessHandler,
 GrapeCity.ActiveReports.Web.v7, Version=7.0.xxxx.0, Culture=neutral,
 PublicKeyToken=cc496777c49a3ff" />
 <add verb="*" path="*.ActiveReport"
 type="GrapeCity.ActiveReports.Web.Handlers.CompiledReportHandler,
 GrapeCity.ActiveReports.Web.v7, Version=7.0.xxxx.0, Culture=neutral,
 PublicKeyToken=cc496777c49a3ff" />
 <add verb="*" path="*.rpx"
 type="GrapeCity.ActiveReports.Web.Handlers.RpxHandler,
 GrapeCity.ActiveReports.Web.v7, Version=7.0.xxxx.0, Culture=neutral,
 PublicKeyToken=cc496777c49a3ff" />
 <add verb="*" path="*.rdl,*.rdlx"
 type="GrapeCity.ActiveReports.Web.Handlers.RdlxHandler,
 GrapeCity.ActiveReports.Web.v7, Version=7.0.xxxx.0, Culture=neutral,
 PublicKeyToken=cc496777c49a3ff" />
 <remove verb="*" path="*.asmx" />
 <add verb="*" path="*.asmx" validate="false"
 type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Extensions,
 Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" />
```

```
</httpHandlers>
```

 **Note:** You might need to update the Version and PublicKeyToken values to reflect the current version of ActiveReports Developer installed on your machine. You can find the Version and PublicKeyToken values in the Global Assembly Cache (GAC), C:\Windows\assembly.

4. On MainPage.xaml, go to the **Properties** window and then to the **Events** view.
5. In the list of events, double-click the viewer1\_Loaded event.
6. On MainPage.xaml.cs/vb that opens, add the following code to the viewer1\_Loaded event:  
**To write code in Visual Basic.NET**

### **Visual Basic.NET Code. Paste to MainPage.xaml.vb to the Viewer1\_Loaded event.**

```
Viewer1.PdfPrint = True
```

#### **To write code in C#**

### **C# Code. Paste to MainPage.xaml.cs to the viewer1\_Loaded event.**

```
viewer1.PdfPrint = true;
```

## Print Methods In ActiveReports Developer

ActiveReports Developer 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
- Print methods in SectionDocument or PageDocument
- Print methods in the PrintExtension class

The following code samples illustrate how to access these Print methods.

#### **Viewer.Print method**

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.

#### **To write the code in Visual Basic.NET**

### **Visual Basic.NET code. Add this code INSIDE the LoadCompleted event of the Viewer**

```
Viewer1.Print(True, True, True)
```

#### **To write the code in C#**

### **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.

#### **Section Report**

#### **To write the code in Visual Basic.NET**

### **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)
```

### **To write the code in C#**

#### **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**

### **To write code in Visual Basic.NET**

#### **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)
```

### **To write code in C#**

#### **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);
```

 **Note:** The Print method is similar to PrintExtension.Print. For details see **Print ('Print Method' in the on-line documentation)** methods in the PrintExtension class below.

### **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.

## SectionReport

### To write code in VisualBasic.NET

#### **Visual Basic.NET code. Paste INSIDE an event like Button\_Click.**

```
GrapeCity.ActiveReports.PrintExtension.Print(sectionDocument, True, True)
```

---

### To write code in C#

#### **C# code. Paste INSIDE an event like Button\_Click.**

```
GrapeCity.ActiveReports.PrintExtension.Print(sectionDocument, true, true);
```

---

## PageReport

### To write code in VisualBasic.NET

#### **Visual Basic.NET code. Paste INSIDE an event like Button\_Click.**

```
GrapeCity.ActiveReports.PrintExtension.Print(pageDocument, True, True)
```

---

### To write code in C#

#### **C# code. Paste INSIDE an event like Button\_Click.**

```
GrapeCity.ActiveReports.PrintExtension.Print(pageDocument, true, true);
```

 **Note:** The Print methods above are accessible through the GrapeCity.ActiveReports.Viewer.Win.v7 assembly. Make sure that you add a reference to this assembly and an Imports (Visual Basic.NET) or using (C#) statement for the GrapeCity.ActiveReports namespace in your project.

## 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.

### This section contains information about

#### **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

Learn about the samples provided with the ActiveReports Developer installation standard and professional editions. The samples folder is located at the following location:

*[User Documents folder]\ComponentOne Samples\ActiveReports Developer 7*

Each sample has a **C#** and a **Visual Basic.NET** version for Visual Studio **2008, 2010** and **2012**. You can also see the comments within the sample projects throughout code.

This section contains information about:

#### **Page Report**

This section provides information on each of the Page report sample (CPL and FPL) provided with the ActiveReports

Developer installation.

## ***Section Report***

This section provides information on each of the Section report sample provided with the ActiveReports Developer installation.

## ***Professional***

This section provides information on each of the sample provided with the ActiveReports Developer professional edition installation.

## **Page Report**

In Page Report, the layout has two variations, **Continuous Page Layout (CPL)** and **Fixed Page Layout (FPL)**.

This section provides information on both CPL and FPL samples.

### ***FPL Samples***

Learn about different FPL samples categorized under API, Layout and Data.

### ***CPL Samples***

Learn about different CPL samples categorized under API, Layout and Web.

## **FPL Samples**

Learn about the different FPL samples categorized under API, Data and Layout respectively.

This section contains:

### ***API***

This section contains a sample that showcases working with ActiveReports API.

### ***Data***

This section contains a sample that showcases working with unbound data source.

### ***Layout***

This section contains a sample that showcases creating different report layouts in FPL.

## **API**

Learn about the samples that fall under the API category.

This section contains:

### ***Custom Resource Locator***

This Custom Resource Locator sample showcases a custom implementation of the resource locator to load pictures from the user's "My Pictures" directory.

## **Custom Resource Locator**

The Custom Resource Locator sample demonstrates a custom implementation of the resource locator to load pictures from the user's **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

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Page Reports\FPL\API\VB.NET\CustomResourceLocator
C#
```

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Page Reports\FPL\API\C#\CustomResourceLocator
```

### Runtime Features

When you run this sample, you see the **MainForm** with the list of images from the **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 **My Pictures** directory. If the directory does not contain any pictures, you should add them to the folder manually.

## Project Details

### 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 runtime.

This folder also contains the **NoImage.bmp** image file that is used if there is no image in the **My 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 runtime 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.

## Data

Learn about the sample that fall under the Data category.

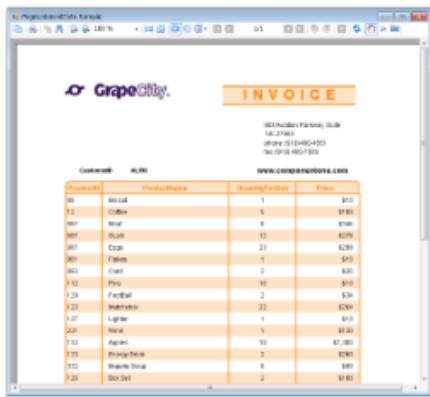
This section contains:

### Page Unbound Data Sample

The PageUnboundData sample demonstrates how to connect a page report to an unbound data source at runtime, using the DataSet provider with the LocateDataSource event.

## Page Unbound Data Sample

The PageUnboundData sample demonstrates how to connect a page report to an unbound data source at runtime, using the DataSet provider with the LocateDataSource event. The reporting engine raises the LocateDataSource event when it needs input on the data to use.



ProductID	ProductName	QuantityOrdered	Price
T01	Small	1	\$10
T02	Coffee	5	\$100
T03	Tea	8	\$80
T04	Book	12	\$20
T05	CD	21	\$20
T06	Flower	1	\$10
T07	Coin	3	\$10
T12	Pepsi	10	\$10
T28	Football	2	\$20
T29	Handmade	32	\$100
T37	Laptop	4	\$10
T40	Wine	1	\$10
T52	Apple	10	\$100
T53	Orange	7	\$100
T54	Musical Chair	8	\$10
T58	Bon Bon	2	\$10

## Sample Location

### Visual Basic.NET

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Page Reports\FPL\Data\VB.NET\PageUnboundDataC#
```

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Page Reports\FPL\Data\C#\PageUnboundData
```

### Runtime 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 runtime using the LocateDataSource event and the DataSet provider.

### Project Details

#### Invoice2.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 runtime. 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.

## Layout

Learn about the samples that showcases different report layouts in FPL.

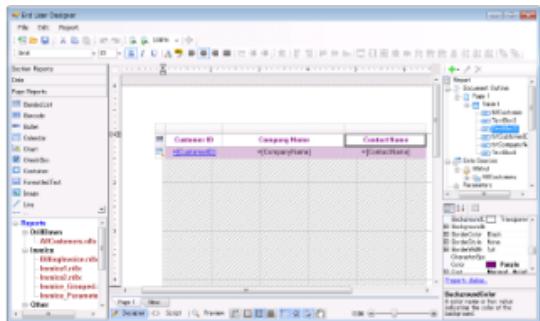
This section contains:

### FPL Report Loader Sample

The FPL Report Loader sample demonstrates how to customize the End User Designer application by adding the TreeView control on the form to display a list of categorized reports.

## FPL Report Loader Sample

The FPL Report Loader sample demonstrates how to customize the End User Designer application by adding the TreeView control on the form to display a list of categorized reports. At runtime, the user can double-click any report to load it into the designer.



## Sample Location

### Visual Basic.NET

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Page Reports\FPL\Layout\VB.NET\FPLReportLoader
C#
```

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Page Reports\FPL\Layout\C#\FPLReportLoader
```

### Runtime 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 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

#### Reports folder

This folder contains several subfolders that are categorized according to the report type.

#### DrillThrough

**AllCustomers report:** This report uses the NWind shared data source connection to provide data. This report 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.

**CustomerDetails report:** This report uses the NWind shared data source connection to provide data. This report is used as a child report for **AllCustomers** report that appears at runtime. It uses the **Table** data region to display the customer orders in the past. It gets displayed when the user clicks any **CustomerID** link in the **AllCustomers** report. The Textbox displaying the **OrderID** field in the detail row of the Table is used to set a drill-through link to navigate to the **OrderDetails** report.

**OrderDetails report:** This report uses the NWind shared data source connection to provide data. This report is used as a child report for **CustomerDetails** report. It uses the **Table** data region to display order details for the **OrderID** value when the **OrderID** link in the **CustomerDetails** report is clicked.

#### Invoice

**BillingInvoice report:** This report uses the NWind shared data source connection to provide data. This report showcases a billing invoice layout commonly used in convenience stores. The report mostly contains Label, TextBox and Line controls in its layout. The **EAN128FNC1** barcode is used in this report due to its high reading accuracy in convenience stores.

**Invoice\_Grouped report:** This report uses the NWind shared data source connection to provide data. This report uses the Table, few TextBox and Label controls to display customer transactions in the Invoice format. The page is grouped on the **CustomerID** field, therefore all transactions made by a customer appear together.

**Invoice\_Parameters report:** This report uses the NWind shared data source connection and two datasets to provide data. This report is similar to the **Invoice\_Grouped** report with an additional feature of parameters. This report uses parameters set on the **CustomerID** field to filter data at report preview.

**Invoice1 report:** This report uses the NWind shared data source connection to provide data. 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. One of the texboxes in the footer section of the BandedList control uses the Sum function to display the GrandTotal of all transactions.

**Invoice2 report:** This report uses the NWind shared data source connection to provide data. This report 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. One of the textboxes placed inside the Container control at the bottom of the report uses the Sum function to display the GrandTotal of all transactions.

#### Other

**Catalog report:** This report uses the NWind shared data source connection to provide data. This report uses multiple page layouts to create a catalog. The layout of this report is spread over to four pages, which appear one after another at runtime. **Page1** and **Page2** layouts simply contain Image and Textbox controls for displaying 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 excessive data. The layout on **Page4** uses Textbox, Shape and Line controls to create a Order Form, which a

user is to fill manually.

**CellMerging report:** This report uses the NWind shared data source connection to provide data. This report demonstrates cell merging in a **Matrix** data region, where cells with same values are merged automatically to delete duplicate values.

**DeliverySlip\_theme.rdlx:** This report uses the Seikyu2 shared data source connection to provide data. This report uses TextBox, Labels, 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, therefore the invoices are sorted by **EstimateID**.

**EmployeeSales report:** This report uses the NWind shared data source connection to provide data. This report uses the Chart and the Table data regions to display sales by each employee for the year of 1997. The Chart uses the Column chart to display a graphical analysis of sales by each employee; the Table lists down the exact numbers. One of the TextBox controls on the report also uses the Sum function to display the grand total of all sales.

**IRS-W4 report:** This report uses the IRS XML data source to provide data to the report. This report uses Textbox, CheckBox, Shape and Line controls to create a simple layout of a form that is used for banking purposes.

**Letter report:** This report uses the NWind shared data source connection to provide data. This report uses 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 order ids with order dates and order amount and is linked with the **OverflowPlaceholder** control to display excessive 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 the **Global** field 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.

**MatrixSample.rdlx:** This report uses the NWind shared data source connection to provide data. This report uses the **Matrix** data region to display an orders report for category name and product name by year and quarter with the totals information. The **Matrix** uses the multiple **row grouping** to group matrix data by **CategoryName** and **ProductName**, and the multiple **column grouping** to group matrix data by **Years** and **Quarter**. One of the TextBox controls on the report also uses the **Sum function** to display the grand total of order amount.

**PurchaseReport report:** This report uses the NWind shared data source connection to provide data. This report uses Textbox and Table controls to display purchase details of each company. The layout page is grouped by the **CompanyName** field to filter data on each report page according to each company. The Textbox control placed below each table column uses the **Sum function** to display total for each table column. The **FixedSize** property of the Table is set to fit in excessive data.

**ReelsMatrix.rdlx:** This report uses the Reels shared data source connection to provide data. It contains the **Matrix** data region to display a sales report for each country, city and media type by year with the totals information. The **Matrix** uses the multiple **row grouping** to group matrix data by **Country**, **City** and **MediaType**, and the **column grouping** to group matrix data by **Years**. One of the TextBox controls on the report also uses the **Sum function** to display the grand total of all orders.

**SalesReport report:** This report uses the Reels OleDb data source connection and two datasets that fetch data by the Stored Procedure query. The layout of the report uses the Chart to display sales and profit for the selected date range and the Table to display chart data. The Table uses the Detail grouping by **EmployeeID** to provide totals of the sales and profit for the selected date range. It also uses the **DataBar** function to plot the profits. 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.

**TackSeal.rdlx:** This report uses the NWind shared data source connection to provide data. It contains the **OverflowPlaceholder** controls 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 the sequence of the data columnar display within the report page.

## ReportsForm

This is the main form that appears when you run this sample. This form uses a ToolStripPanel, ToolStripContentPanel, Designer, Toolbox, ReportExplorer, TreeView and a PropertyGrid control to build a customized End User Designer.

The code-behind of the form contains code that sets the designer and creates a blank page based report. It also contains code that attaches the Toolbox, ReportExplorer and PropertyGrid controls to the Designer, inserts DropDown items to the ToolStripDropDownItem and sets their functions, adds the reports to the TreeView control and loads a report in the designer when double-clicked and checks for any modifications that have been made to the report loaded in the designer.

## CPL Samples

Learn about the different CPL samples categorized under API, Layouts and Web respectively.

This section contains:

### API

This section contains samples that showcase working with ActiveReports API to create a CPL report.

### Layouts

This section contains a sample that showcases creating different report layouts in CPL.

## **Web**

This section contains a sample that showcases displaying a CPL report at client side.

## **API**

Learn about the samples that fall under the API category.

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.

### **DataSet DataSource**

This sample demonstrates how to use a dataset as a data source for a report.

### **NormalizedDataSet**

This sample demonstrates loading the same data into three types of reports using the DataSet provider, XML provider and Object provider for fetching data from the data source.

### **OleDbDataSource**

This sample demonstrates how to connect to an OleDb data source at runtime and pass data to the report using LocateDataSource event.

### **ReportWizard**

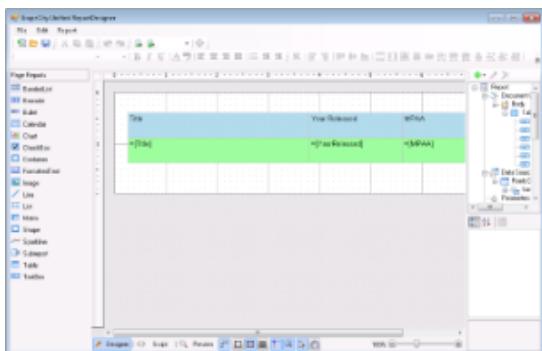
This sample demonstrates how to create a custom Report Wizard that allows you to select a report from the list of multiple reports and then allows you to select the data that you want to display in the selected report.

### **XMLDataProvider**

This sample demonstrates how to connect to a XML data source at runtime and pass data to the report using LocateDataSource event.

## **Create Report**

The Create Report sample demonstrates how to create a CPL report using code and display it in the ActiveReports Designer.



### **Sample Location**

#### **Visual Basic.NET**

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Page Reports\CPL\API\VB.NET\CreateReport
C#
```

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Page Reports\CPL\API\C#\CreateReport
```

#### **Runtime Features**

When you run this sample, the ActiveReports Designer appears with a CPL 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**. A copy is located at **[User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data**. If you are unable to access the data files in the sample, create the Data folder and place all the data files to this folder, or change the reference path to the data files within the folder according to your environment.

## Project Details

### UnifiedDesignerForm

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 runtime. 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

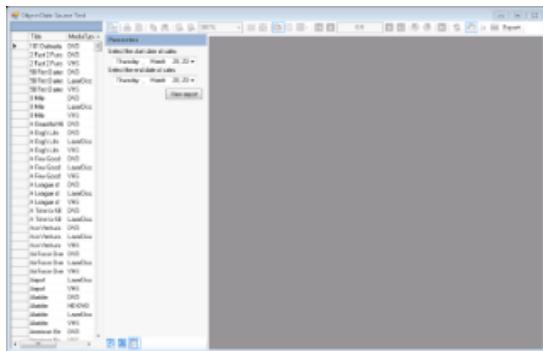
This file is an internal class that contains string values that are required for creating a dataset of the report.

### LayoutBuilder

This file is an internal class that contains code for creating a CPL report layout and adding a data source and a dataset to it.

## DataSet DataSource

The DataSetDataSource sample demonstrates how to use the API to load the report and to set the report's dataset at runtime.



### Sample Location

#### Visual Basic.NET

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Page Reports\CPL\API\VB.NET\DataSetDataSource
C#
```

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Page Reports\CPL\API\C#\DataSetSource
```

### Runtime Features

When you run this sample, the **MainForm** with the ActiveReports Viewer appears. The Viewer displays the Parameter pane and the **DataGridView** control containing the report data in the left part of the Viewer. Set the parameters and click the View Report button to display the report.

Also, the Viewer toolbar contains the custom export button that opens the **ExportProperties** form where you can set up the options for exporting the displayed report to a specified format.

## Project Details

### Constants

This file is an internal class that contains string values that are required for creating a dataset of the report.

### DataLayer

This file is an internal class that contains code to provide data for the report.

### ExportForm

This is the form that appears when you click the custom **Export** button on the Viewer's toolbar at runtime. In the Export form, you can select from the following export file format options.

- HTML
- Excel
- Text
- PDF
- RTF
- TIFF

This form uses the ComboBox to show the supported export formats, the Button control to browse to the location where you want to save the exported file, the PropertiesGrid to list down the properties of the selected exported format and two other Button controls that allow you to export the report to a file and reset the fields respectively.

Right-click the form and select **View Code** to see how to export the report document and define the respective function of Browse, Export and Reset buttons in their Click event.

## MainForm

The MainForm has the **Splitter** control with the ActiveReports **Viewer** control docked to the right and the **DataGrid** control docked to the left. The Viewer control displays the report at runtime while the DataGrid control lists the report's data.

Right-click the form and select **View Code** to see how to initialize and return data for the report, show and load the report in the Viewer and add data to the DataGrid control. It also contains code that adds the custom export button to the Viewer's toolbar and defines its function in the **exportbutton\_Click** event.

## ReportFromDataSet

This is the main report that the ActiveReports Viewer displays. This report features the **TextBox**, **Table** and **Subreport** controls. The TextBox and Table controls display data from the dataset; the Subreport control links to the **SubReportFomDataSet** report.

The report has two parameters, **StartDate** and **EndDate**. When the report is displayed in the Viewer, you are prompted to select the start date and end date of sales on the Parameters pane of the Viewer sidebar.

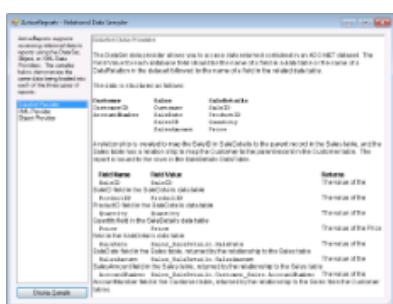
## SubReportFomDataSet

This report gets displayed by the SubReport control of the **ReportFromDataSet** report. This report features the **Table** data region that displays data from the dataset.

The report has two parameters, **StartDate** and **EndDate**.

## Normalized DataSet

The Normalized DataSet sample demonstrates how to access relational data in reports, using the DataSet, Object or XML providers. This sample uses the same data in all three reports bound to the relational data providers.



## Sample Location

### Visual Basic.NET

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Page Reports\CPL\API\VB.NET\NormalizedDataSet
C#
```

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Page Reports\CPL\API\C#\NormalizedDataSet
```

### Runtime Features

When you run this sample, the **Relational Data Sampler** form appears. Select a provider in the left pane and click the **Display Sample** button located below the left pane to see a sample report appear in the **Unified ReportDesigner** form. You can see the description of the each provider and its related sample report as you select the provider in the left pane of the form.

### Project Details

#### Explanations folder

This folder contains **DataSetProvider.rtf**, **ObjectDataProvider.rtf** and **XmlDataProvider.rtf** files that provide descriptions for the corresponding

provider and its sample report. This is the description that the **Relational Data Sampler** form displays when you run the sample and select a provider.

## Constants

This file is an internal class that contains string values required for creating a dataset for the report. It also contains the sample description text that the label on the main form displays.

## DataLayer

This file is an internal class that contains code to read data from the Reels database and load it in a normalized dataset.

## DataSetReport

This is the sample report that is bound to the DataSet provider. This report uses an ADO.Net dataset to obtain data for the report. At runtime, the report is displayed in **Unified ReportDesigner** after you select **DataSet Provider** in the list of providers on the main **ReportSelector** form and click the **Display Sample** button.

## ObjectReport

This is the sample report that is bound to the Object provider. This report uses a collection of business objects to obtain data for the report. At runtime, the report is displayed in **Unified ReportDesigner** after you select **Object Provider** in the list of providers on the main **ReportSelector** form and click the **Display Sample** button.

## XmlReport

This is the sample report that is bound to the XML provider. This report uses the XmlReader to obtain data for the report. At runtime, the report is displayed in **Unified ReportDesigner** after you select **XML Provider** in the list of providers on the main **Relational Data Sampler** form and click the **Display Sample** button.

## XmlDB

This is an XML database file included in this sample to provide data for the reports.

## ReportSelector

This is the main form that is displayed at runtime. This form uses the Label, ListBox and Button controls docked to the left and the RichTextBox control docked to the right. The Label control displays description of the sample, the Listbox control displays the list of three data providers at runtime. The Button control is used to display the corresponding sample report in the ActiveReports Designer.

Right-click the form and select **View Code** to see how to add the three sample reports in an array to display in the listbox and have the ActiveReports Designer display the report when you double-click the provider in the list or when you click the **Display Sample** button.

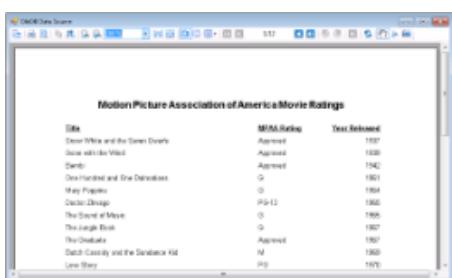
## UnifiedDesignerForm

This form appears when you click the **Display Sample** button on the main **Relational Data Sampler** form after selecting a data provider in the listbox. This form uses the ToolStripPanel, the ToolStripContentPanel, the Designer, the Toolbox, the ReportExplorer and 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, the ReportExplorer and the PropertyGrid controls to the Designer, adds DropDownList items to the ToolStripDropDownItem and sets their functions and checks for any modifications to the report loaded in the **Unified ReportDesigner**.

## OleDb DataSource

The OleDb Data Source sample demonstrates how to use the OleDb data provider for binding a report to data.



## Sample Location

## Visual Basic.NET

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Page Reports\CPL\API\VB.NET\OleDbDataSourceC#
```

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Page Reports\CPL\API\C#\OleDbDataSource
```

Runtime 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**. A copy is located at **[User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data**. If you are unable to access the data files in the sample, create the Data folder and place all the data files to this folder, or change the reference path to the data files within the folder according to your environment.

## 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 runtime.

Right-click the form and select **View Code** to see how to load and show the report at runtime.

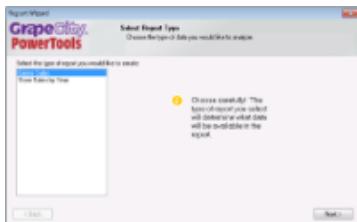
### OleDbReport.rdlx

This is the report that gets displayed in the Viewer at runtime.

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.

## Report Wizard

The Report Wizard sample demonstrates how to create and customize a report, using the report wizard.



### Sample Location

#### Visual Basic.NET

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Page Reports\CPL\API\VB.NET\ReportWizard
C#
```

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Page Reports\CPL\API\C#\ReportWizard
```

#### Runtime Features

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.

## Project Details

### 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 runtime.

#### 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 are summarizable.

#### 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

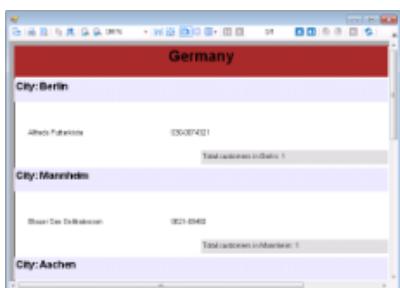
This is the internal class file that contains code for handling the UI of the Report Wizard.

#### 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.

## Xml Data Provider

The XML Data Provider sample demonstrates how to use the XML data provider for supplying data to the report.



## Sample Location

### Visual Basic.NET

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Page Reports\CPL\API\VB.NET\XmlDataProvider
C#
```

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Page Reports\CPL\API\C#\XmlDataProvider
```

### Runtime 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 runtime. 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.

#### DataLayer

This is the internal class file that contains code to provide data for the reports in this sample.

#### MainForm

This is the main form that appears when you run this sample. It uses the **ActiveReports Viewer** control to display the report at runtime.

Right-click the form and select **View Code** to see how to set different properties of the Viewer control and the form. It also contains code to load and show the report at runtime.

#### MyXmlDB.xml

This XML file is a database that contains data that is fetched by the DataLayer class for supplying it to the reports.

## Layouts

Learn about the samples that showcases different report layouts in CPL.

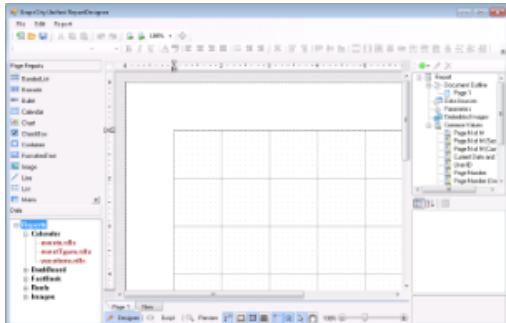
This section contains:

### CPLReportLoader

This sample demonstrates how to create a standalone designer with an additional feature of a report loader that provides a tree view of several reports that you can directly load in the designer by double-clicking them.

### CPL Report Loader

The CPL Report Loader sample demonstrates how to customize the GrapeCity Unified ReportDesigner application by adding the TreeView control to the form to display a list of categorized reports. At runtime, you can double-click any report to load it into the designer.



## Sample Location

### Visual Basic.NET

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Page Reports\CPL\Layouts\VB.NET\CPLReportLoader
C#
```

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Page Reports\CPL\Layouts\C#\CPLReportLoader
```

### Runtime Features

When you run this sample, the GrapeCity Unified ReportDesigner appears with a list of report categories at the bottom left of the form. Expand each category to view 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

#### Reports folder

This folder contain subfolders that are named according to each report category. Each report category contains several reports.

#### Calendar folder

##### events.rdlx

This report uses the **Calendar** control to display events.

##### eventsTypes.rdlx

This report uses the **Calendar** control to display Family, Music and Cultural events and the **Container** controls to set a custom event background color for each event type.

##### vacations.rdlx

This report uses the **Calendar** control to display employee's vacations.

#### DashBoard

##### CallCenterDashBoard.rdlx

This report 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.rdlx

This report uses 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.rdlx

This report uses **drill-through links** to navigate to the **TeamStatisticsDashboard.rdlx** report that displays the selected team statistics.

##### TeamStatisticsDashboard.rdlx

This report uses the **Bullet** control to display the team's scoring performance in ranges below average, average and above average. The report also uses the **Sparkline** controls to display the team statistics.

#### FactBook

##### CountryFacts.rdlx

This report uses the **List** data region that groups data by **Country**. The report also contains the map image, which **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** country.

The dataset has a filter that retrieves only countries, specified by the report parameter.

By each textbox with category under **Energy Production / Consumption** you can see the **Image** control that uses the **Icon Set** function to flag energy categories where consumption is greater than production.

##### LifeExpectancyByGdpAndMedianAge.rdlx

This report uses the **Matrix** data region to compare the average life expectancy based on the category where the median age and GDP fall. The Median Age and GDP categories are calculated fields that use the **Choose** and **Switch** functions respectively.

##### Top10CountriesByGDP.rdlx

This report contains the **Image** control with a database image that uses the **Range Bar** function to create an ad-hoc horizontal bar chart.

#### Reels

##### CustomerMailingList.rdlx

This report uses the **Container** control with the TextBox and Barcode controls. The **Columns** property of the Body section is set to **3** to create mailing labels.

##### CustomerOrders.rdlx

This report uses 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 has its **Value** property set to Sum function to calculate the total with tax for each order as well as the grand 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.

## **CustomerOrdersCoupon.rdlx**

This report demonstrates how to create a coupon where the amount of coupons is determined by the report **parameter**.

The Reels logo displayed on the report is embedded within the Reels **theme**.

## **DistrictReport.rdlx**

This report uses the **Chart** and **Matrix** data regions to display the number of sold items and the profit for each month of the two year span (2004-2005) for the selected district.

The report **parameter** determines what district to display and uses the available values from the second report dataset **SalesData**.

The Reels logo that is displayed on the report is embedded within the Reels **theme**.

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.rdlx**

This report uses the **BandedList** data region that groups data by **SaleYear**, **DistrictID** and **RegionID** to display district sales details.

The Reels logo that is displayed on the report is embedded within the Reels **theme**.

## **Filmography.rdlx**

This report uses the nested **List** data regions that group data by **MovieID** and **MoviePersonID** to return a distinct number of movies with the selected actor(s).

The report contains two **cascading parameters** that narrow down the number of actors displayed in the list.

The Reels logo that is displayed on the report is embedded within the Reels **theme**.

## **GenreSales.rdlx**

This report uses the **Matrix** data region to display the units sold for each genre, each year and each quarter. The report also uses the **plain column Chart** to display the number of titles sold for each genre.

This report uses the multivalue **parameter** that allows you to select more than one genre for displaying sales data.

The Reels logo that is displayed on the report is embedded within the Reels **theme**.

## **GenreStatistics.rdlx**

This report uses the Median and Mode **aggregate functions** to show the middle value 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.

## **MonthlySalesReport.rdlx**

This report uses the **Table** data region that groups data by **District**. 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**.

## **MovieReport.rdlx**

This report uses 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 report Body uses the **drill-through link** (the **Action** is set to **Jump to report**) that opens the **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**.

## **ReelsConfidential.rdlx-master**

This **master report** is a template for other reports. This master report provides the connection to the **Reels** data source that child reports can use.

The Reels logo that is displayed on the report is embedded within the Reels **theme**.

## RegionPerformance.rdlx

This report uses the **Table** data region to display data. The Table 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. The Sum Textbox controls in the Table have their **Value** properties set to **expressions** 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.rdlx

This report uses the **Table** data region without grouping. The Table detail row has its **BackgroundColor** property set to the **expression** to create a green bar report.

The Reels logo that is displayed on the report is embedded within the Reels **theme**.

## SalesByMediaType.rdlx

This report uses the **List** data region to display data. The List data is grouped by **Image**.

The List contains 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**.

This report uses the **ReelsConfidential.rdlx-master** report to render the report page header and footer.

## SalesByRegion.rdlx

This report uses the **Matrix** data region and **subtotals** to display the number of units sold and profit for each region by year and quarter. The Matrix report data has the row grouping by **Region** and the column grouping by **SaleDate.Year**.

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**.

The image in the report PageHeader uses the **embedded images** from a local image folder.

## SalesReceipt.rdlx

This report uses the **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** textboxes 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.rdlx

This report uses the **Table** and **Chart** data regions to display totals of sales and profit for the selected date range. The Table also uses the **DataBar 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.rdlx

This report identifies stores with profits above or below expectations. The report 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.rdlx

This report uses the **Table** and **Chart** data regions to display sales and profit for each employee.

The Table and Chart data is grouped by **EmployeeID**. The Table uses the 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 space of 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 initially displayed. By clicking the toggle image next to the supervisor's name, the rows with details about employees are displayed.

The report uses the 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.rdlx

This report uses 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 are 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.

## UnifiedDesignerForm

This is the main form that appears when you run this sample. This form uses the ToolStripPanel, ToolStripContentPanel, Designer, Toolbox, ReportExplorer, TreeView and PropertyGrid controls to create a customized 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 DropDownList 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.

# Web

Learn to display a CPL report at client side.

This section contains:

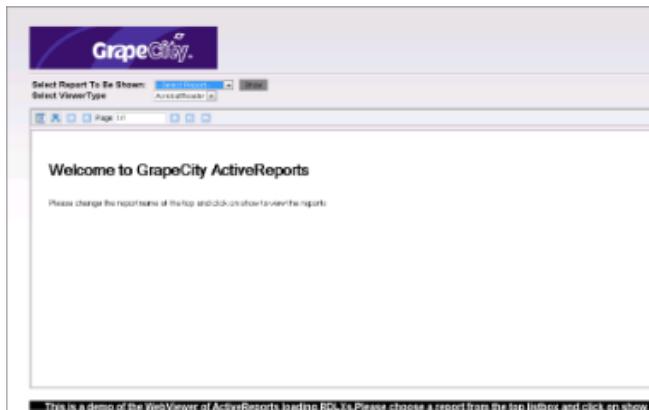
## PageReportsOnWeb

This sample demonstrates the use of ASP.NET features, including HTTP Handlers, Report Caching, and the Server Viewer Control in both Visual Studio 2008 and 2010.

 **Note:** The ASP.NET user machine must have ASP.NET write access before you run the sample or an exception is thrown during execution.

## Page Reports On Web

The Page Reports On Web sample demonstrates how to use the ActiveReports WebViewer control on the ASPX page to display reports. The sample offers two drop-down lists of CPL reports and WebViewer types.



## Sample Location

### Visual Basic.NET

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Page Reports\CPL\Web\VB.NET\PageReportsOnWeb
C#
```

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Page Reports\CPL\Web\C#\PageReportsOnWeb
```

### Runtime Features

When you run this sample, a webpage with two drop-down lists, one with CPL reports and the other with viewer types, appears. Select a

report and the viewer type from the drop-down lists and click the **Show** button to see the report appear in the selected viewer type.

## Project Details

### Images folder

This folder contains two image files - **bg.jpg** and **grapecity\_logo.jpg**, that are displayed on the Default.aspx page at runtime.

### ActiveReports.ReportService.asmx

The report web service required for the proper functioning of the Web Viewer. The ActiveReports.ReportService.asmx is added automatically to the project when you place the WebViewer control on the form. You can also add this service by using **Add New Item** in the Visual Studio 2010 **Project** menu.

For the information on how to use the WebViewer control, see **Getting Started with the Web Viewer**.

### Default.aspx

This is the main page that appears at runtime. This ASPX page contains drop-down lists to collect user input, a button to send the collected values to the WebViewer control.

Right-click the ASPX page and select **View Code** to see the code used to find the reports in the **Reels** folder installed along with the sample. To learn more about the CPL reports that you can select on this page, see **CPL Report Loader**. It also contains code that displays the selected report and sets the WebViewer type selected by the user.

The **Source** tab of the ASPX page contains the marquee tag to display a floating message below the WebViewer.

### ErrorSelection.rdlx

This is the report that appears at runtime when you have clicked the **Show** button without selecting any specific report from the drop-down list. It contains a TextBox control that displays the error message.

### Main.rdlx

This is the default report that appears in the WebViewer when you run this sample. It contains a **FormattedText** control with the **HTML** property set to the welcome message text. This text is displayed when you run this sample.

### Web.config

The configuration file that contains the httpHandlers that allow ActiveReports Developer to process reports on the Web.

Note that you need to manually update version information here when you update your version of ActiveReports Developer.

## Section Report

Learn about the different section report samples categorized under Data, Layout, Preview, Summary and Web respectively.

This section contains:

### Data

This section contains samples that showcase working with different data sources.

### Layout

This section contains samples that showcase creating different layouts in a section report.

### Preview

This section contains samples that showcase creating an interactive section report during preview.

### Summary

This section contains samples that showcase displaying summarized data in a section report.

### Web

This section contains a sample that showcases displaying a section report at client side.

## Data

Learn working with different data sources.

This section contains:

### Bound Data Sample

Demonstrates binding to ADO.NET Data objects.

## **IList Binding 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 Sample**

The LINQ sample demonstrates how to use LINQ in an ActiveReports Developer report.

## **Unbound Data Sample**

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.

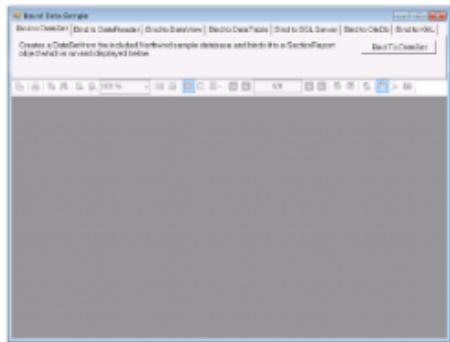
## **XML Sample**

This sample demonstrates how to create a report with XML data, using a SubReport or using the XML hierarchical structure.

## Bound Data Sample

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 seven 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

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\VB.NET\Data\BoundData
C#
```

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\C#\Data\BoundData
```

#### Runtime Features

The top panel of the MainForm comprises of seven 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.

## 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 seven 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 **Grouping Data in Section Reports**.

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.

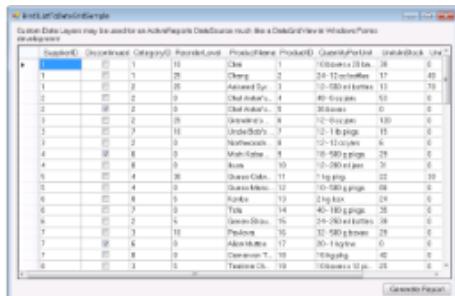
#### PageFooter

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**.

## IList Binding Sample

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

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\VB.NET\Data\IListBinding
C#
```

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\C#\Data\IListBinding
Runtime 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**. A copy is located at **[User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data**. If you are unable to access the data files in the sample, create the Data folder and place all the data files to this folder, or change the reference path to the data files within the folder according to your environment.

## Project Details

The IList Binding sample consists of two projects: **IListBinding** and **IListBinding.DataLayer**.

### IListBinding Project

#### BindIListToDataGridViewSample

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

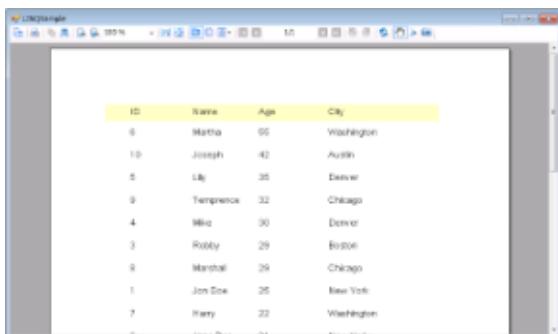
Defines custom collection class.

### 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 Sample

The LINQ sample demonstrates how to use LINQ in an ActiveReports Developer report.



## Sample Location

### Visual Basic.NET

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\VB.NET\Data\LINQ
C#
```

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\C#\Data\LINQ
Runtime 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.

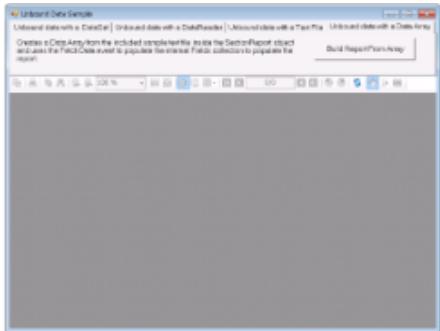
### rptLINQToObject report

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 Sample

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

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\VB.NET\Data\UnboundData
C#
```

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\C#\Data\UnboundData
Runtime 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 Sample

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

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\VB.NET\Data\xml
C#
```

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\C#\Data\xml
```

#### Runtime 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.

# Layout

Learn to create different layouts in a section report.

This section contains:

### ***Annual Report Sample***

Demonstrates how to use subreports and nested subreports, how to modify data source properties at run time, how to use parameters in the chart control, how to create alternate row highlighting and how to use the page break control.

### ***Category Selection Sample***

Demonstrates using the ad hoc report filter by modifying the report's SQL query at run time.

### ***Charting Sample***

Demonstrates the use of the Chart control in both bound and unbound modes.

### ***Cross Section Control Sample***

Demonstrates the use of the new cross section lines and boxes.

### ***Cross Tab Report Sample***

Demonstrates using unbound data, conditional highlighting and distributing data across columns to create a cross-tab view and data aggregation.

### ***Inheritance Sample***

Demonstrates using the method that inherits a report at runtime and design time.

### ***Layout Loader Sample***

Demonstrates using the MDI window to load multiple report layouts.

### ***Style Sheets Sample***

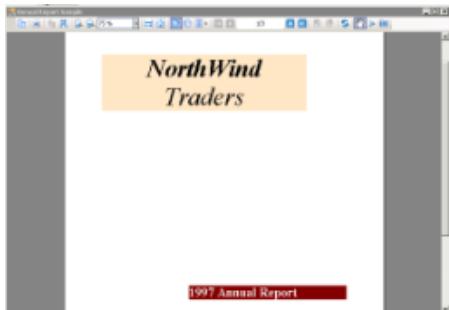
Demonstrates changing styles at run time to provide a different look to a same report.

### ***Subreport Sample***

Demonstrates using subreports in an ActiveReports Developer report.

## Annual Report Sample

The AnnualReport sample demonstrates the use of SubReports, section report properties, and Chart control. It includes a StartupForm and AnnualReport, ProductSalesByCategory, Top10, Top10Customers, Top10Products reports.



### Sample Location

#### Visual Basic.NET

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\VB.NET\Layout\AnnualReport
C#
```

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\C#\Layout\AnnualReport
Runtime features
```

When you run this sample, a three page report appears wherein the first page mentions the name of the trader and year of the report, the second page displays company information along with a SubReport displaying category wise sales and the third page displays data in form of charts. The AnnualReport fetches data from ProductSalesByCategory report and Top10 report and displays it in the Viewer control. The Top10 report further fetches data from Top10Customers subreport and Top10Products subreport.

**Note:** To run this sample, you must have access to the **Nwind.mdb**. A copy is located at **[User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data\NWIND.mdb**. If you are unable to access the data files in the sample, create the Data folder and place all the data files to this folder, or change the reference path to the data files within the folder according to your environment.

### Project Details

#### StartupForm

This form contains the ActiveReports Developer **Viewer** control. The **Dock** property of the viewer is set to **Fill** so that it resizes automatically with the form at run time. Right-click and select **View Code** to see the code that displays the AnnualReport when the form loads.

#### AnnualReport report

This is the main report for the project.

#### ReportHeader section

This report features a two-page **ReportHeader** section that uses a **PageBreak** control to separate the two pages, and breaks to the second page by setting the ReportHeader section's **NewPage** property to **After**. This report shows how you can use the **BackColor** and **ForeColor** properties of labels to create a distinctive look for your reports.

The ReportHeader section also has a **SubReport** control that links to the ProductSalesByCategory report in the code behind the report, in the **ReportStart** event.

**Tip:** It is recommended that you initialize reports in the **ReportStart** event rather than a section **Format** event so that a new report is not initialized every time the section runs. If the SubReport control is in a section that prints multiple times, you need to assign the report in the section Format event while still initializing in the ReportStart event.

The yellow background in the right half of the ReportHeader section below the page break control is achieved by using the **Shape** control and setting its **BackColor** property. The image to the left is a **Picture** control.

#### Detail section

The Detail section contains one **SubReport** control that link in the code behind the report to the **Top10** reports. In most reports, the Detail section would run multiple times. In this report, the Detail section has only labels, and no bound textboxes, so it will only run once. Therefore, the Top10 report can be assigned to the SubReport control in the **ReportStart** event where it is initialized.

Notice that the **ReportFooter** section has its **Height** property set to **0**. This is because, except for the Detail section, all sections come in

pairs. In order to use the ReportHeader section, you must also have a ReportFooter section. If you do not want to use it, you can set its Height to 0.

### ProductSalesByCategory report

This is the report that is assigned to the **SubReport** control in the ReportHeader section of the Annual Report.

 **Tip:** This report has the ReportHeader/Footer and PageHeader/Footer sections removed. These sections are not printed within the SubReport control, so removing the sections saves processing time.

Notice also that the **PrintWidth** property of this report is only **2.677** inches. This is so that it fits easily within the SubReport control on the Annual Report.

This report uses the **GroupHeader** section to display labels for the data fields that fill the **Detail** section. The fields in the Detail section repeat once for each row of data in the database.

Right-click in the grey area around the report and select **View Code** to see the code that sets the data source for the report. Notice that the background color of the Detail section is set to yellow on every second row of data.

### Top10 report

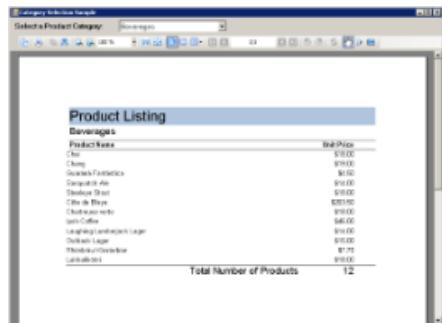
This report only has a Detail section. The Detail section contains two SubReport controls that linked to **Top10Customers** and **Top10Products** reports in code.

#### Top10Customers and Top10Products reports

The Top10Customers and Top10Products reports use only two sections, GroupHeader and Detail. The **PrintWidth** property of each report is set to **3.135** inches so that it fits into the SubReport control in the Top10 report. The GroupHeader section of each report is filled with a **Chart** control. The Detail section of each report has two bound **TextBoxes** and a **Label** controls. Right-click and select **View Code** to see the code that sets the data source for the report, passes data to the Chart control, alternates the background color of the Detail section, and sets the **Text** property of the label.

## Category Selection Sample

The CategorySelection sample demonstrates passing a SQL string into a report at run time. It consists of a CategorySelectForm and CategoryProducts report.



### Sample Location

#### Visual Basic.NET

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\VB.NET\Layout\CategorySelection
C#
```

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\C#\Layout\CategorySelection
```

#### Runtime Features

When you run this sample, a form containing a Viewer control and a ComboBox control within a panel is displayed. The **Select a Product Category** ComboBox at the top of the form allows the users to select the type of data they want to display in the Viewer control.

 **Note:** To run this sample, you must have access to the **Nwind.mdb**. A copy is located at **[User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data\NWind.mdb**. If you are unable to access the data files in the sample, create the Data folder and place all the data files to this folder, or change the reference path to the data files within the folder according to your environment.

### Project Details

#### CategorySelectForm

The Viewer control fills most of the form, and a ComboBox at the top allows the user to select which data to send to the Viewer control.

To see how all of this works, right-click the form and select **View Code**.

- The **getCategories** code populates the ComboBox with the category names.
- The **runCategoryReport** code runs the report with a SQL string with the CategoryName passed in by the ComboBox selection.
- The **SelectIndexChanged** event calls runCategoryReport and passes the CategoryName.

## CategoryProducts

This report lists the products in the selected category, and summarizes the number of products.

### ReportHeader section

This section cannot be deleted, because the related ReportFooter section is used and thus the **Height** property is set to **0**.

### PageHeader section

This section uses **Label** controls, Line controls and a TextBox. **DataField** property of **txtCategory** TextBox is set to **CategoryName**. Two **Line** controls and three **Label** controls create the report title and the column headers for this report.

### Detail section

This section contains two bound **TextBox** controls to display each product in the selected category along with its price. Although the form passes data to the report at run time, the report's data source is set for design time use. At design time, it is easier to drag bound fields onto the report from the Report Explorer than it is to create them and set their properties. The data source also allows you to preview the report while you are designing it.

### PageFooter section

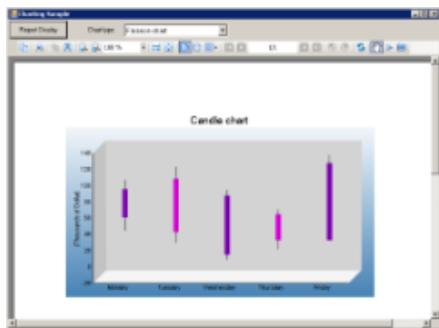
This section uses the **ReportInfo** control to display Page N of M. In the Properties window, you can select a way to display the page number and run date using the **FormatString** property.

### ReportFooter section

This section contains a **Label** control and a bound **TextBox**. The TextBox has the **SummaryType** set to **GrandTotal** and **DataField** property set to **ProductName** to display the total number of products in the selected category.

## Charting Sample

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

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\VB.NET\Layout\Charting
C#
```

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\C#\Layout\Charting
Runtime Features
```

#### 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.mdb** database. A copy is located at **[User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data\NWIND.mdb**. If you are unable to access the data files in the sample, create the Data folder and place all the data files to this folder, or change the reference path to the data files within the folder according to your environment.

## Project Details

### 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.mbd** 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

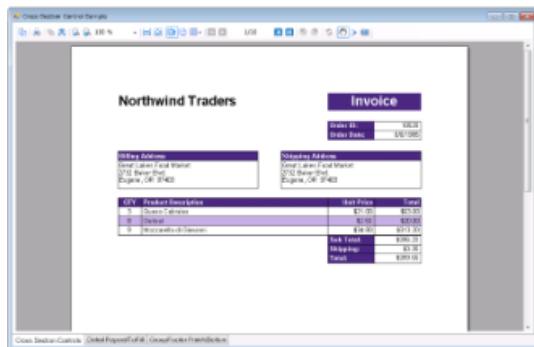
Displays candle chart on a report. Chart data is set at design time using **DataPoint Collection Editor**. DataSource property is not used for this chart.

### 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 Sample

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 Location

#### Visual Basic.NET

<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\VB.NET\Layout\CrossSectionControls  
C#

<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\C#\Layout\CrossSectionControls  
**Runtime Features**

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**. A copy is located at **[User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data\NWIND.mdb**. If you are unable to access the data files in the sample, create the Data folder and place all the data files to this folder, or change the reference path to the data files within the folder according to your environment.

## Project Details

### ViewerForm

The ViewerForm has three tabs-**Cross Section Controls**, **Detail RepeatToFill**, **GroupFooter PrintAtBottom**, each with an ActiveReports Developer 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 section

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 Sample

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**

<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\VB.NET\Layout\CrossTabSample C#

<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\C#\Layout\CrossTabSample  
**Runtime Features**

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**. A copy is located at **[User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data\NWIND.mdb**. If you are unable to access the data files in the sample, create the Data folder and place all the data files to this folder, or change the reference path to the data files within the folder according to your environment.

### Project Details

## 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.

## **Product Weekly Sales 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 **txtPQTDCChange** 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 **txtPQTDCChange** 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

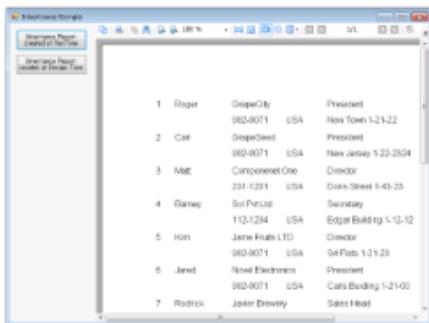
This section is not used, so it is hidden by setting the **Height** property to **0** and/or **Visible** property to **False**. Otherwise, it would print at the bottom of each page. The section cannot be deleted, because the related PageHeader section is in use.

## 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.

## Inheritance Sample

This sample explains the method to inherit a report at runtime and design time. The Inheritance sample solution comprises of two classes - the parent class and the child class for both inheritance at runtime and design time.



## Sample Location

### Visual Basic.NET

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\VB.NET\Layout\Inheritance C#
```

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\C#\Layout\Inheritance Runtime Features
```

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 runtime 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.

## Project Details

### 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.

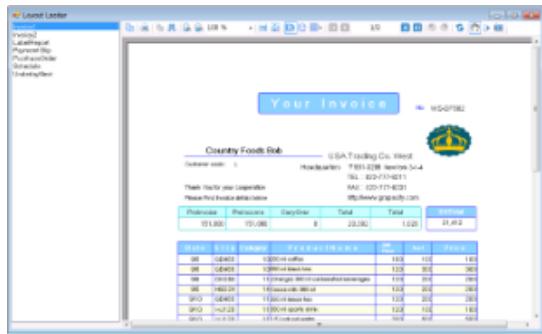
### rptInheritChild

The designer that sets layout for each control and its DataField property.

## Layout Loader Sample

The Layout Loader sample demonstrates the use of main form with the Viewer and the Windows ListBox control that displays a list of reports from the sample Reports folder. The Layout Loader sample summarizes the following seven sample reports in a single sample.

- Invoice1
- Invoice2
- LabelReport
- PaymentSlip
- PurchaseOrder
- Schedule
- UnderlayNext



## Sample Location

### Visual Basic.NET

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\VB.NET\Layout\LayoutLoader
C#
```

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\C#\Layout\LayoutLoader
Runtime Features
```

When you run this sample, the main form appears, with the list of the sample reports in the Windows ListBox control on the left of the form. Double-click the report from the list to have it displayed in the Viewer control on the right of the form.

**Note:** To run this sample, you must have access to the **Shiire.mdb**, **BILL.mdb**, **Nwind.mdb**, **Seikyu2.mdb** and **Schedule.mdb**. A copy is located at **[User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data**. If you are unable to access the data files in the sample, create the Data folder and place all the data files to this folder, or change the reference path to the data files within the folder according to your environment.

## Project Details

### ViewerForm

This is the main form that displays the list of reports in the Windows ListBox control on the left of the form. This form also contains the Viewer control that displays a report once it is selected in the list.

### Reports

The **Reports** folder contains .rpx reports that are displayed on the main ViewerForm when the sample project is run.

### Invoice1 report

The report calculates the invoice 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.

**Note:** To run this sample, you must have access to the **BILL.mdb**. A copy is located at **[User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data**. If you are unable to access the data files in the sample, create the above folder to place all the data files or change the reference path to the data files within the folder according to your environment.

### 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, we do not recommend that you place bound fields on the PageHeader or PageFooter sections because these sections are displayed on a page once. For the same reason we do not recommend that you place bound fields on the GroupHeader and GroupFooter sections.

## ghColumnCaption section

This section contains the **Label** controls that are captions for the information displayed in the Detail section.

This section also uses the **CrossSectionLine** and the **CrossSectionBox** controls that span the ghColumnCaption section to end in the Detail section, forming vertical lines between columns of the invoice details and a rectangle around the details of the invoice at run time.

## Detail section

This section contains the bound **TextBox** controls that control 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

When you run this sample, one set of the two pages report is displayed in the Viewer control.

 **Note:** To run this sample, you must have access to **Seikyu2.mdb**. A copy is located at **[User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data**. If you are unable to access the data files in the sample, create the above folder to place all the data files or change the reference path to the data files within the folder according to your environment.

## 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 end in the GroupFooter1 section, forming 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** as 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 report

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.

 **Note:** To run this sample, you must have access to the **Nwind.mdb**. A copy is located at **[User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data**. If you are unable to access the data files in the sample, create the above folder to place all the data files or change the reference path to the data files within the folder according to your environment.

## 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 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.

In general, GS1-128 is used for payment purposes by convenience stores that require high accuracy. However, the reading accuracy of the barcode may be reduced due to the printer resolution, the page size or the ink. To avoid this issue, we use the EAN128FNC1 barcode for the dotted correction of blackbar in this Sample report.

 **Note:** To run this sample, you must have access to the **BILL.mdb**. A copy is located at **[User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data**. If you are unable to access the data files in the sample, create the above folder to place all the data files or change the reference path to the data files within the folder according to your environment.

## Detail section

This section contains the 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 the purchase slip. Purchase slips are created based on the bound data. This report 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. The calculated fields are used to calculate the total cost and the total selling price.

 **Note:** To run this sample, you must have access to the **Shiire.mdb**. A copy is located at **[User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data**. If you are unable to access the data files in the sample, create the above folder to place all the data files or change the reference path to the data files within the folder according to your environment.

## 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 the **Textbox** controls. It uses the 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.

 **Note:** To run this sample, you must have access to **Schedule.mdb**. A copy is located at **[User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data**. If you are unable to access the data files in the sample, create the above folder to place all the data files or change the reference path to the data files within the folder according to your environment.

## 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** control and the **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.

To display multiple reports within the Detail section, the Date data is compared with previous records in the **DetailFormat** event and if the dates match, only the record pointer is moved by dynamically switching the **LayoutAction** property without changing the output positioning of Detail.

### UnderlayNext report

The report displays the list of customers based on the bound data.

 **Note:** To run this sample, you must have access to **Nwind.mdb**. A copy is located at **[User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data**. If you are unable to access the data files in the sample, create the above folder to place all the data files or change the reference path to the data files within the folder according to your environment.

## PageHeader section

This section contains the **Label** and **Line** controls to create the header for the report data.

## GroupHeader1 section

This section contains the bound Textbox control, **Country**, and the CrossSectionLine controls to create the table for the report body data.

The **UnderlayNext** property 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

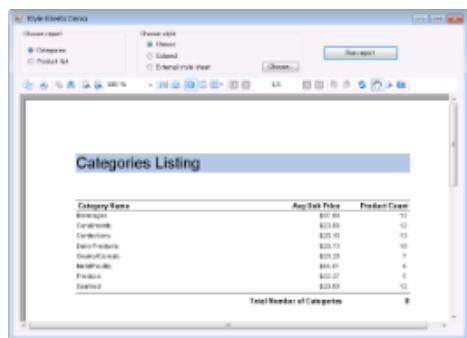
This section contains the bound **Textbox** controls to display the report data.

## GroupFooter1 section

This section contains a **Line** control to separate each data group in the report body.

# Style Sheets Sample

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 reportstyles and a form containing the ActiveReports Viewer control and other controls that allow you to select any combination of styles and reports.



Category Name	Avg Unit Price	Product Count
Beverages	\$11.99	10
Canned Beverage	\$11.99	10
Condiments	\$11.99	10
Deli Products	\$11.99	10
Dishwasher Detergent	\$11.99	10
Fresh Fruits	\$11.99	10
Genérico	\$11.99	10
Health Foods	\$11.99	10
International	\$11.99	10
Milk Products	\$11.99	10
North Beach	\$11.99	10
Specialty Foods	\$11.99	10
Total Number of Categories		8

## Sample Location

### Visual Basic.NET

<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\VB.NET\Layout\Stylesheets C#

<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\C#\Layout\stylesheet

### Runtime Features

#### 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.

### Project Details

#### 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

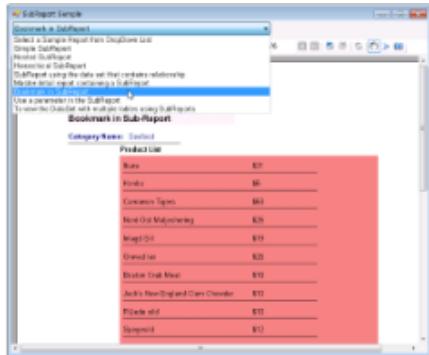
This event contains code that sets up an Open dialog that shows only \*.reportstyle files, and passes the selected reportstyle path and file name string to the externalStyleSheet variable.

#### 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.

## Subreport Sample

The SubReports sample demonstrates how SubReport control can be used to generate nested and hierarchical reports.



### Sample Location

#### Visual Basic.NET

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\VB.NET\Layout\SubReport
C#
```

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\C#\Layout\SubReport
```

### Runtime Features

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. See rptSimpleMain

and `rptSimpleSub` for details.

- **Nested SubReport** - the report demonstrates how to nest subreports to display main, child, and grandchild levels in a report. See `rptNestedParent`, `rptNestedChildMain` and `rptNestedChildSub` for details.
- **Hierarchical SubReport** - the main report dataset with the **SHAPE** statement defines the hierarchical structure of the report that uses a subreport. See `rptHierarchicalMain` and `rptHierarchicalSub` for details.
- **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. See `rptDSRelationParent`, `rptDSRelationChildMain` and `rptDSRelationChildSub` for details.
- **Master-detail report containing a SubReport** - the sample report that demonstrates how to create a master detail report that uses a subreport. See `rptMasterMain` and `rptMasterSub` for details.
- **Bookmark in SubReport** - the sample report that uses bookmarks from the subreport. See `rptBookmarkMain` and `rptBookmarkSub` for details.
- **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**. A copy is located at **[User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data\NWIND.mdb**. If you are unable to access the data files in the sample, create the Data folder and place all the data files to this folder, or change the reference path to the data files within the folder according to your environment.

## Project Details

### ViewerForm

The ViewerForm uses the ActiveReports **Viewer** control in the bottom section of the form, and a panel docked to the top with the drop-down list. The Viewer displays a report once it is selected from the list.

### rptBookmarkMain

This report is displayed when you select the **Bookmark in SubReport** option in the drop-down list of the ViewerForm.

This report uses bookmarks that you can see in the **Document map** of the Viewer sidebar, which is displayed if you click the **Toggle sidebar** button in the Viewer's toolbar. See **Linking in Reports** and **Add Bookmarks** for details on using bookmarks in ActiveReports Developer.

The report uses the PageHeader section to display the name of the report, the Detail section to display data and the PageFooter section to display the report information. The Detail section contains the **Subreport** control that displays data from `rptBookmarkSub`.

The report is bound to **Nwind.mdb** at runtime.

### rptBookmarkSub

This report contains the Product List information for the Subreport control in `rptBookmarkMain`.

This report also contains bookmarks, which are setup in code in the **Detail Format** event.

### rptDSRelationChildMain

This report displays the Product Name information and is the subreport for `rptDSRelationParent`.

The report uses the GroupHeader section to group data by Product Name and to display the label for the data in the Detail section, and the Detail section to display data. The Detail section contains the **Subreport** control that displays data from `rptDSRelationChildSub`.

The report is bound to the data row collection.

### rptDSRelationChildSub

This report displays the Order Details information and is the subreport for `rptDSRelationChildMain`.

### rptDSRelationParent

This report is displayed when you select the **SubReport using the data set that contains relationship** option in the drop-down list of the ViewerForm. This report demonstrates how to bind a subreport to a dataset with a relation, defined in the **DataSet.Relations** property in code, and how to work with nested data relations.

The report uses the PageHeader section to display the name of the report, the groupHeader1 section to provide labels for the report data, the ghCategories to group data by Category Name, the Detail section to display data, and the PageFooter section to display the

report information. The Detail section contains the **Subreport** control that displays the Product Name data from **rptDSRelationChildMain**.

The report is bound to the data row collection.

### **rptHierarchicalMain**

This report is displayed when you select the **Hierarchical SubReport** option in the drop-down list of the ViewerForm.

The report uses the Detail section to display data and the PageFooter section to display the report information. The Detail section contains the **Subreport** control that displays data from **rptHierarchicalSub**.

The report is bound to Nwind.mdb at runtime.

### **rptHierarchicalSub**

This report displays the Order information and is the subreport for **rptHierarchicalMain**.

### **rptMasterMain**

This report is displayed when you select the **Master-detail report containing a SubReport** option in the drop-down list of the ViewerForm. This report displays orders with the general information, the order details in this report are taken from **rptMasterSub**.

The report uses the PageHeader section to display the name of the report, the Detail section to display data and the PageFooter section to display the report information. The Detail section contains the bound Textbox controls that display information for each order and the **Subreport** control that displays order details from **rptMasterSub**.

The report is bound to Nwind.mdb at runtime.

### **rptMasterSub**

This report displays the Order Details information and is the subreport for **rptMasterMain**.

### **rptNestedChildMain**

This report displays the Order Details information and is the subreport for **rptNestedParent**.

The report uses the groupHeader1 section to group the data by Orders ID, the groupHeader2 section to display static labels for the data in the Detail section, and the Detail section to display data. The groupHeader2 section contains the **Subreport** control that displays data from **rptNestedChildSub**.

The report is bound to Nwind.mdb at runtime.

### **rptNestedChildSub**

This report displays the Contact information and is the subreport for **rptNestedChildMain**.

The report is bound to Nwind.mdb at runtime.

### **rptNestedParent**

This report is displayed when you select the **Nested SubReport** option in the drop-down list of the ViewerForm. This report demonstrates how to set up embedded subreports to display main, child, and grandchild levels in a report.

The report uses the PageHeader section to display the name of the report, the groupHeader section to group the report data by Employee ID and to display static labels for the data in the Detail section, the Detail section that displays data on each employee, and the PageFooter section to display the report information. The Detail section contains the **Subreport** control that displays data from **rptNestedChildMain**.

The report is bound to Nwind.mdb at runtime.

### **rptParamMain**

This report is displayed when you select the **Use a parameter in the SubReport** option in the drop-down list of the ViewerForm. This report demonstrates how to set up a parameter in the subreport. See **Parameters** for more details.

The report uses the PageHeader section to display the name of the report, the groupHeader section to group data by Country and to display static labels for the report data, the Detail section to display Contact information, and the PageFooter section to display the report information. The Detail section contains the **Subreport** control that displays data from **rptParamSub**.

The report is bound to Nwind.mdb at runtime.

### **rptParamSub**

This report displays the Product Name information and is the subreport for **rptParamMain**.

The report is bound to Nwind.mdb at runtime.

The parameter is added to the sql query at runtime. Right-click **rptParamSub** in the Solution Explorer and select **View Code** to see

the code implementation for the subreport parameter.

See **Add Parameters in a Section Report** for more details.

### **rptSimpleMain**

This report is displayed when you select the **Simple SubReport** option in the drop-down list of the ViewerForm.

The report uses the PageHeader section to display the name of the report, the Detail section and the PageFooter section to display the report information. The Detail section contains the bound Textbox control to display the Category Name information and the **Subreport** control to display data from **rptSimpleSub**.

The report is bound to Nwind.mdb at runtime.

### **rptSimpleSub**

This report displays the Product Name information and is the subreport for **rptSimpleMain**.

The Detail section contains the **Barcode** control.

### **rptUnboundDSMain**

This report is displayed when you select the **To view the DataSet with multiple tables using SubReports** option in the drop-down list of the ViewerForm.

The report uses the PageHeader section to display the name of the report, the Detail section and the PageFooter section to display the report information. The Detail section contains the bound Textbox controls to display Customer ID and Company Name, and the **Subreport** control to display data from **rptUnboundDSSub**.

The report is bound to the DataSet with multiple tables at runtime. Right-click **rptUnboundDSMain** in the Solution Explorer and select **View Code** to see the code implementation for the dataset connection.

### **rptUnboundDSSub**

This report displays the Order Detail information and is the subreport for **rptUnboundDSMain**.

The report uses the GroupHeader section to display static labels for Order List details in the Detail section.

## Preview

Learn to create an interactive section report during preview.

This section contains:

### **Custom Annotation Sample**

Demonstrates adding the Custom Annotation button to the report Viewer toolbar and adding a new annotation to the report.

### **Custom Preview Sample**

Demonstrates using Viewer control customization, export filters, rich edit control and mail-merge, parameters in the chart control, and grouping.

### **Hyperlinks and Drill Down Sample**

Demonstrates using hyperlinks and the viewer hyperlink event to simulate drill-down from one report to another.

### **Print Multiple Pages Per Sheet Sample**

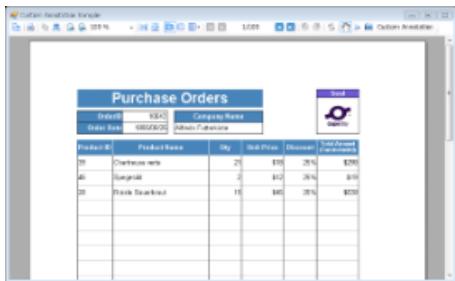
Demonstrates printing a document with multiple pages per sheet by using the common PrintDocument class of the .NET.Framework.

### **RDF Viewer Sample**

Demonstrates creating the Rdf Viewer with an ActiveReports Developer Viewer control and an ExportForm with a Property Grid.

## Custom Annotation Sample

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

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\VB.NET\Preview\CustomAnnotation
C#
```

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\C#\Preview\CustomAnnotation
```

### Runtime Features

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. A copy is located at [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data. If you are unable to access the data files in the sample, create the Data folder and place all the data files to this folder, or change the reference path to the data files within the folder according to your environment.

## Project Details

### 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

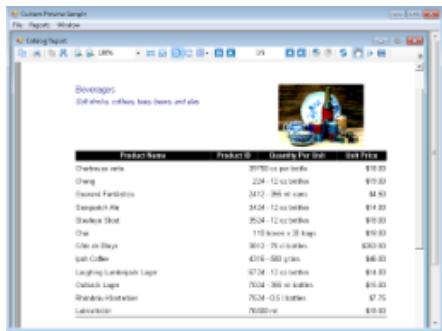
This folder holds the icon used for adding annotation (seal image) to the report.

### Resource1.resx

This file contains the string for the message box that appears when **Custom Annotation** button is clicked again to add the annotation.

## Custom Preview Sample

The Custom Preview sample demonstrates how to create various type of reports, customize the Viewer toolbar, load or save a report document file (RDF). The sample also provides six types of export options and demonstrates the use of print settings.



## Sample Location

### Visual Basic.NET

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\VB.NET\Preview\CustomPreview
C#
```

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\C#\Preview\CustomPreview
Runtime Features
```

The Custom Preview sample consists of the parent CustomPreviewForm with menus, the child PreviewForm with the Viewer control, the ExportForm with the Properties window, the Reports folder, and the Resources folder.

When you run the sample, the CustomPreviewForm appears. It has three menus - **File**, **Reports** and **Window**.

The **File** menu allows you to perform the following operations.

- **Open:** Loads the report document file (RDF).
- **SpreadBuilder API:** Creates an Excel file by the Spreadbuilder API.
- **Export...** This option is available with an opened report. Exports a report to six formats - HTML, Excel, Text, PDF, RTF, and TIFF.
- **Save:** This option is available with an opened report. Saves the opened report in the report document format (RDF).
- **Exit:** Closes the Custom Preview Sample form.

The **Reports** menu allows opening the sample reports by selecting them from the list. The available reports are described in detail below under Reports folder.

The **Window** menu allows navigating between the opened reports.

**Note:** To run this sample, you must have access to the **Nwind.mdb**. A copy is located at **[User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data**. If you are unable to access the data files in the sample, create the above folder to place all the data files or change the reference path to the data files within the folder according to your environment.

## Project Details

### CustomPreviewForm

This is the main form that gets displayed when you run the sample.

The CustomPreviewForm has its **IsMdiContainer** property set to **True**. This ensures that the child PreviewForm is embedded into the parent CustomPreviewForm.

This form has a menu bar, **mnuMain**, with three menus: **File**, **Reports**, and **Window**. The **MergeType** property of the File menu is set to **MergeItems**, which adds the menu items from any child PreviewForms to the CustomPreviewForm. The form also has two dialogs: **dlgOpenFile**, and **dlgPrint**.

Right-click the form and select **View Code** to see how reports selected from the Reports menu are run and passed to the PreviewForm, using the **ShowReport** method.

### PreviewForm

This is the form that gets displayed when you select a report in the **Reports** menu of the main Custom Preview Sample form (CustomPreviewForm).

The PreviewForm has the ActiveReports Viewer control with its **Dock** property set to **Fill**. This ensures that the viewer resizes along with the form at run time.

The form has the File menu with its **MergeType** set to **MergeItems**. This ensures that the File menu on the CustomPreviewForm displays the **Export**, **Save** menu items when a report is open. The form also has one dialog: **dlgPrint**.

Right-click the form and select **View Code** to see how two custom buttons are added to the toolbar in the **PreviewForm\_Load** event. The **ToolClick** event of the Viewer calls the SaveDocument and ExportDocument functions for the new buttons. The menu item click events call

the same functions for the related menu items.

The **SaveDocument** function opens the dialog **dlgSave**, while the **ExportDocument** function opens the new **ExportForm**.

## ExportForm

This is the form that gets displayed when you select **Export** in the **File** menu of the opened report. In the Export form, you can select from the following export file format options.

- HTML
- Excel
- Text
- PDF
- RTF
- TIFF

The Export Report Document form opens by the **ExportDocument** function on the **PreviewForm**. The form features the Export Format combo box, **cboExportFormat**, that populates the **PropertyGrid** control based on the selected item. The export types are added to **cboExportFormat** via the **Items (collection)** property.

Right-click the form and select **View Code** to see how the property grid control's **SelectedObject** is set to the selected export in the **cboExportFormat SelectedIndexChanged** event. This ensures that only the properties related to each export type are displayed in the Properties window.

See the **btnOK Click** event for the code that exports the report to the selected file name and format, and the **btnSaveFile Click** event for the code that opens the Save dialog.

## Resources folder

This folder contains the icons that are used in the File menu of the Custom Preview Sample form.

## Reports folder

The Reports folder contains the following reports that demonstrate the Viewer and Export features.

### Catalog

The Catalog report uses the **PageBreak** control and the **NewPage** property to create a cover at the beginning and an order form at the end of the catalog. It uses grouping to list products by Category.

#### ReportHeader section

At the top of the ReportHeader section, the **Picture**, **Line**, and **Label** controls are used to create a static cover page for the catalog. The **PageBreak** control allows a second page of static labels to be set in the same section. Setting the section's **NewPage** property to **After** ensures that the report breaks to a new page before rendering the next section. The ReportHeader section prints once per report.

#### PageHeader section

This section is not in use, so the **Height** property is set to **0**. This section cannot be deleted because its related **PageFooter** section is in use.

#### ghCategoryName section

This GroupHeader section has its **DataField** property set to **CategoryName**. This, along with sorting the data by the same field, ensures that all details for one category are printed before the report moves on to the next category. Also, the section's **GroupKeepTogether** property is set to **All**. This causes ActiveReports to attempt to print the group header, related details, and the group footer together on one page.

The controls in this section include two bound **TextBox** controls and a bound **Picture** control, along with a row of labels to serve as column headers for the Detail section to follow.

#### Detail section

Click the gray report **DataSource** icon on the Detail section band to open the **Report Data Source** dialog, where you can see the Connection String and SQL Query that provide data for the bound fields.

The Detail section has four bound **TextBox** controls. Select a textbox and go to the **DataField** property in the Properties window to see a bound data field for this textbox. The **UnitPrice** textbox also uses the **OutputFormat** to display the data in the currency format. This section prints once for each row of data.

#### gfCategoryName section

This section is used only to render a horizontal Line control after each category grouping is completed.

#### PageFooter section

This section is used to display the document title and the page number at the bottom of each page.

#### ReportFooter section

This section has the **NewPage** property set to **Before** to ensure that it begins at the top of a new page.

The **Label**, **Shape**, and **Line** controls are used to create the static order form layout in this page-long report section that prints once at the end of the report.

## CustomerLabels

### Detail section

Click the gray report DataSource icon on the Detail section band to open the **Report Data Source** dialog, where you can see the Connection String and SQL Query that provide data for the bound fields.

The Detail section has bound **TextBox** and **Label** controls. Select a textbox and go to the **DataField** property in the Properties window to see a bound data field for this textbox.

The **ColumnCount** property is set to 2, which allows the report to display the labels in 2 columns. The **ColumnDirection** property is set to **AcrossDown** to have labels print in the left-to-right order instead of the top-to-bottom order.

## EmployeeProfiles

### PageHeader section

This section is used to display the document title and the text description at the top of each page.

### Detail section

Click the gray report DataSource icon on the Detail section band to open the **Report Data Source** dialog, where you can see the Connection String and SQL Query that provide data for the bound fields.

The Detail section has a bound **Picture** control, bound **TextBox** and **Label** controls. Select a textbox and go to the **DataField** property in the Properties window to see a bound data field for this textbox.

This report uses bookmarks that you can see in the **Document map** of the Viewer sidebar, which is displayed if you click the **Toggle sidebar** button in the Viewer's toolbar. Right-click the report and select **View Code** to see how to use the unbound Textbox control and the **Add.Bookmark** method in the **Detail\_BeforePrint** event to set the bookmarks in this report. See **Linking in Reports** and **Add Bookmarks** for details on using bookmarks in ActiveReports Developer.

The **KeepTogether** property is set to **True** to print the section content on a single page.

### PageFooter section

This section is not in use, so the **Height** property is set to 0. This section cannot be deleted because its related **PageHeader** section is in use.

## EmployeeSales

### ReportHeader section

This section contains Label controls, a bound Textbox control and a bound Chart control to display the title, total sales and the bar graph representation of the data.

### GroupHeader1 section

The controls in this section include two Label controls to serve as column headers for the Detail section to follow.

### Detail section

Click the gray report DataSource icon on the Detail section band to open the **Report Data Source** dialog, where you can see the Connection String and SQL Query that provide data for the bound fields.

The Detail section has bound **TextBox** and **Label** controls. Select a textbox and go to the **DataField** property in the Properties window to see a bound data field for this textbox.

### GroupFooter1 section

This section is not in use, so the Height property is set to 0. This section cannot be deleted because its related GroupHeader1 section is in use.

### Report Footer section

This section is not in use, so the Height property is set to 0. This section cannot be deleted because its related ReportHeader section is in use.

## Invoice

### PageHeader section

This section contains a Picture control with a logo, 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.

The **KeepTogether** property is set to **True** to print the section content on a single page.

### Detail section

Click the gray report DataSource icon on the Detail section band to open the **Report Data Source** dialog, where you can see the Connection String and SQL Query that provide data for the bound fields.

This section contains bound **TextBox** controls. These controls render once for each row of data found in the current OrderID before the report moves on to the Footer sections.

### GOrderID 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. Go to the **Script** tab at the bottom of the report to see how this TextBox is set.

The **KeepTogether** property is set to **True** to print the section content on a single page.

### PageFooter section

This section has one simple **Label** control that contains a "Thank you" note. For more information about report sections and the order in which they print, see [Section Report Structure](#) and [Section Report Events](#).

### Letter

#### PageHeader section

This section contains the Picture control with a logo, the Line control, and the Label controls.

The **CanGrow** property is set to **False** to indicate that the controls are to be clipped to the section's height.

#### ghCustomerID section

This section has its **DataField** property set to **CustomerID**. This ensures that all details for one customer id are printed before the report moves on to the next customer id.

The section contains the **Label** controls, the **RichTextBox** control, and bound **TextBox** controls.

The **RichTextBox** control allows to create a mail-merge report where field place holders are replaced with values (merged) at run time. Right-click the report and select **View Code** to see how to use the **ReplaceField** method in the **BeforePrint** event to calculate total fields in the **RichTextBox** control.

The **CanShrink** property is set to **True** to have the section shrink to fit its controls.

The **KeepTogether** property is set to **True** to print the section content on a single page.

### Detail section

Click the gray report DataSource icon on the Detail section band to open the **Report Data Source** dialog, where you can see the Connection String and SQL Query that provide data for the bound fields.

This section contains three bound TextBox controls.

The **KeepTogether** property is set to **True** to print the section content on a single page.

#### gfCustomerID section

This section contains two **Label** controls that display the note at the end of each letter to a customer.

The **NewPage** property is set to **After** to ensure that the report breaks to a new page before rendering the next section.

### PageFooter section

This section contains the **Label** control that displays the address of the company.

The **CanGrow** property is set to **False** to indicate that the controls are to be clipped to the section's height.

## Hyperlinks and DrillThrough Sample

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

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\VB.NET\Preview\Hyperlinks and DrillThrough
C#
```

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\C#\Preview\Hyperlinks and DrillThrough
```

### Runtime Features

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 access to the **Nwind.mdb**. A copy is located at **[User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data\Nwind.mdb**. If you are unable to access the data files in the sample, create the Data folder and place all the data files to this folder, or change the reference path to the data files within the folder according to your environment.

## Project Details

### ViewerForm

This form contains only the Viewer control with 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 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.

### DrillThroughMain Report

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.

**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.

#### 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.

#### 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 DrillThroguh1.

### GroupHeader section

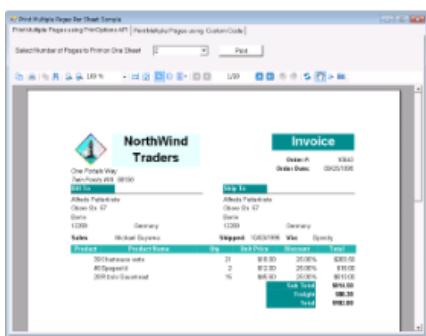
Like in the previous report, this section contains Label controls to serve as column headers for the details, and a CrossSectionBox control.

### 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.

## Print Mutliple Pages per Sheet Sample

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

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\VB.NET\Preview\PrintMultiplePagesPerSheet
C#
```

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\C#\Preview\PrintMultiplePagesPerSheet
```

#### Runtime features

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 Number 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**. A copy is located at **[User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data\Nwind.mdb**. If you are unable to access the data files in the sample, create the Data folder and place all the data files to this folder, or change the reference path to the data files within the folder according to your environment.

### Project details

#### PrintMultiplePagesForm

This form contains the ActiveReports Developer **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.

#### PageHeader section

This section contains a **Picture** control to display the logo along with several **Label** and **TextBox** controls to display company name, order date, address, Order number etc.

### ghOrderID section

The **DataField** property of this section is 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 TextBox controls, as well as two **Line** controls.

### Detail section

This section contains bound TextBox controls. These render once for each row of data found in the current OrderID before the report moves on to the Footer 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.

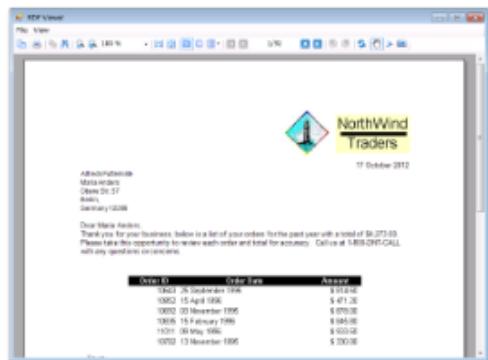
This section contains several Label and TextBox controls. **Subtotal** and **Freight** TextBox use the following properties to summarize the detail data: **SummaryFunc**, **SummaryGroup**, and **SummaryType**. However, the **Total** TextBox does not use the DataField property or any of the summary properties, or even any code. To find the functionality of this TextBox, in the design view, click the **Script** tab at the bottom of the report.

### PageFooter section

This section has one simple Label control. The **CanGrow** property is set to **False**.

## RDF Viewer Sample

The RDF Viewer sample is used to view RDF files. It consists of an **RdfViewerForm** with an ActiveReports Viewer control and an **ExportForm** with a Property Grid. The sample also contains a RDFs folder of saved reports.



### Sample Location

#### Visual Basic.NET

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\VB.NET\Preview\RDFViewer
C#
```

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\C#\Preview\RDFViewer
Runtime features
```

When you run the sample, a Viewer control containing **File** and **View** menu appears on the top. Following options are available in given menus.

#### File menu

- Open: Loads any of the six given RDF files
- Export: Exports file to HTML, Excel, Text, PDF, RTF or TIFF format
- Print: Prints the loaded RDF file
- Exit: Closes the application

#### View menu

- TocView: Shows or hides all the report pages as thumbnails in the left pane
- Toolbar: Shows or hides Viewer toolbar

#### Project Details

#### ExportForm

This form contains a Combobox to collect the user-selected export format, a property grid to display properties for the selected format, and an OK button to export the report to the selected format. It also contains a Save dialog. Right-click the form and select **View Code** to see how this is done.

The **Export Format** ComboBox's **SelectedIndexChanged** event sets the **cmbExportFormat** variable to the selected export type.

The **cmbExportFormat** variable is picked up in the OK button **Click** event, and then the report Document is pulled from the Viewer and exported to the selected format.

### RdfViewerForm

This form contains an ActiveReports Developer Viewer control with its **Dock** property set to **Fill**. This ensures that the viewer resizes along with the form at run time. It also contains a **MenuStrip** and an **OpenFileDialog**. To see the code implementation of this form, right-click the form and select **View Code**.

The **Click** event of the Open option in the File menu filters to show only RDF files and opens the **Open** File dialog to the RDFs folder. The **dlgOpenFile\_FileOK** event loads the selected RDF file into the Viewer control. The **Click** event of the **Export** option in the File menu opens a new ExportForm. For more information on the Viewer control, see **Using the Viewer**.

### RDFs folder

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. For more information, see **Save and Load RDF Report Files**.

The following five reports are included in this sample:

- Catalog.rdf
- Customer Labels.rdf
- Employee Profiles.rdf
- Employee Sales.rdf
- Invoice.rdf
- Letter.rdf

## Summary

Learn to display summarized data in a section report.

This section contains:

### **Calculated Fields Sample**

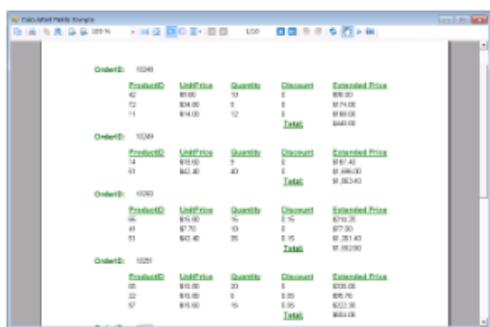
Demonstrates using an unbound data field to perform a calculation which can then be aggregated.

### **Data Field Expressions Sample**

Demonstrates the use of expressions in the **DataField** properties of controls.

## Calculated Fields Sample

The CalculatedFields sample demonstrates the use of calculated fields in a report, 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.



### Sample Location

#### Visual Basic.NET

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\VB.NET\Summary\CalculatedFields
C#
```

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\C#\Summary\CustomAnnotation
```

#### Runtime Features

When you run the sample, a report displaying ProductID, UnitPrice, Quantity, Discount, Extended Price and Total value for each OrderID is

displayed in the Viewer control. The **Extended Price** value is a calculated field that displays the result of the formula specified in FetchData event.

 **Note:** To run this sample, you must have access to the **Nwind.mdb**. A copy is located at **[User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data\NWIND.mdb**. If you are unable to access the data files in the sample, create the Data folder and place all the data files to this folder, or change the reference path to the data files within the folder according to your environment.

## Project Details

### 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 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

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.

## Data Field Expressions Sample

The DataFieldExpressions sample 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 Developer 7.



### Sample Location

#### Visual Basic.NET

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\VB.NET\Summary\DataFieldExpressions
C#
```

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\C#\Summary\DataFieldExpressions
```

#### Runtime Features

When you run the sample, the DataFieldExpressionsReport appears in the Viewer control. 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.

## Project Details

### StartForm

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.

### DataFieldExpressionsReport

The DataFieldExpressionsReport is an unbound report that uses the field values from the OrderDetail file for displaying the report data.

### PageHeader section

This section contains one Label control with the note text and four labels with the names of the report data fields.

### Detail section

This 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

This file contains the class with 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 **DataInitialize** event and the data is bound to the field values in the **FetchData** event of the DataFieldExpressionsReport.

## Web

Learn to display a section report at client side.

This section contains:

### Standard Edition Web Sample

Demonstrates exporting an ActiveReports Developer report to the HTML or PDF format in your Web application.

## Standard Edition Web Sample

The Standard Edition Web 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

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\VB.NET\Web\ActiveReports7WebStd
C#
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Section Reports\C#\Web\ActiveReports7WebStd
```

## Runtime Features

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 access to the **Nwind.mdb**. A copy is located at **[User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data\NWIND.mdb**. If you are unable to access the data files in the sample, create the Data folder and place all the data files to this folder, or change the reference path to the data files within the folder according to your environment.

## Project Details

### 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 **Grouping Data in Section Reports**.

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

The NwindLabels 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.

#### ActiveReports.ReportService.asmx

The report web service required for the proper functioning of the Web Viewer. The ActiveReports.ReportService.asmx is added automatically to the project when you place the WebViewer control on the form. You can also add this service by using **Add New Item** in the Visual Studio **Project** menu.

For the information on how to use the WebViewer control, see **Getting Started with the Web Viewer**.

## 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 Developer to process reports on the Web.

Note that you need to manually update version information here when you update your version of ActiveReports Developer.

# Professional

This section provides information on each of the sample provided with the ActiveReports Developer professional edition installation.

This section contains:

## Custom Data Provider Sample

This sample demonstrates how to create a project using custom data provider and how to pull data from a comma separated value (CSV) file.

## Digital Signature Sample

This sample demonstrates how you can digitally sign or set time stamp for a section report when exporting it to PDF format.

## End User Designer Sample

Demonstrates a custom end-user report designer that can be integrated in your applications to allow users to modify report layouts.

## Professional Web Sample

The ASP.NET Web Samples demonstrate the use of Professional Edition ASP.NET features, including HTTP Handlers, Report Caching, and the Server Viewer Control in both Visual Studio 2008 and 2010.

## Silverlight Viewer Sample

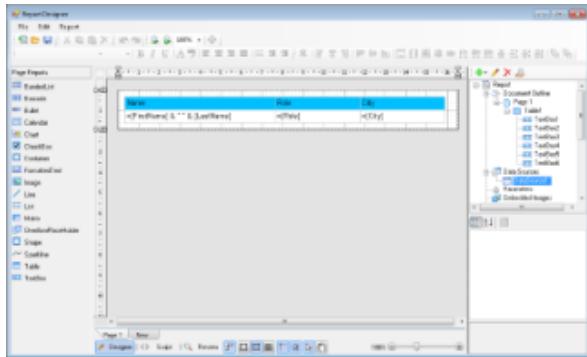
This sample demonstrates the Silverlight Viewer with its various options of loading RPX, RDLX and RDF reports.

## ActiveReports 7 with MVC Sample

The ActiveReports 7 with MVC Sample demonstrates an MVC web application using the ActiveReports Web Viewer with the options of loading section and page report layouts.

## CustomDataProvider

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 Developer Professional Edition.



### Sample Location

#### Visual Basic.NET

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Professional\VB.NET\CustomDataProvider
C#
```

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Professional\C#\CustomDataProvider
```

#### Runtime Features

When you run this sample, a **DesignerForm** displaying **DemoReport.rdlx** report in ActiveReports Designer 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.

#### Project Details

##### CustomDataProvider folder

##### CSVDataProvider folder

This folder contains the following classes.

- 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 folder

##### 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 folder

##### 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.

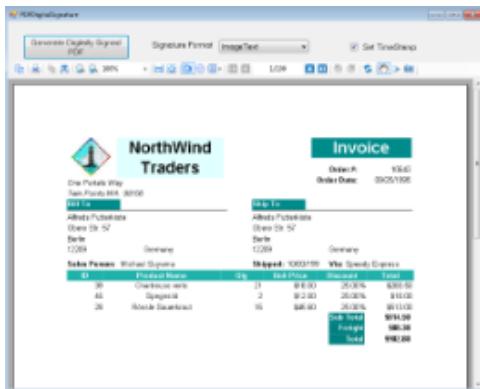
## GrapeCity.ActiveReports.config

The configuration file that configures the project to use the custom data provider.

## Digital Signature Sample

This sample demonstrates how you can digitally sign or set time stamp for a section report when exporting it to PDF format.

 **Note:** PDF digital signatures 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

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Professional\VB.NET\DigitalSignature
C#
```

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Professional\C#\DigitalSignature
```

### Runtime Features

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.mdb**. A copy is located at **[User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data\Nwind.mdb**. If you are unable to access the data files in the sample, create the Data folder and place all the data files to this folder, or change the reference path to the data files within the folder according to your environment.

## Project Details

### 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 <http://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.

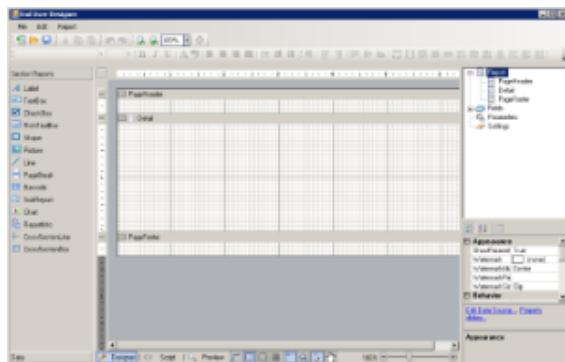
Right-click **PDFDigitalSignature** in the Solution Explorer and select **View Code** to see the code implementation for the pdf digital signature options.

### Resource1.resx

This file contains the string for the message box that appears after the PDF file is generated.

## End User Designer Sample

The EndUserDesigner Sample demonstrates how to set up a custom end-user report designer using the Designer, ReportExplorer and ToolBox controls. This Sample is part of the ActiveReports Developer Professional Edition.



### Sample Location

### Visual Basic.NET

<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Professional\VB.NET\EndUserDesigner  
C#

## Runtime Features

When you run the sample, the End User Designer appears in the Viewer control. This report designer provides the full functionality of the ActiveReports Designer and supports two types of report layouts: **page layout** and **section layout**.

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 and the Report Explorer with a tree view of report controls.

The End User Designer provides the following menu items.

### File menu

- **New** - creates a new report in the designer with the default report template.
- **Open** - opens an **Open File Dialog** to load a report.
- **Export** - exports a report to one of the available formats, such as PDF, HTML, TIFF, Excel, Text, or RTF. This menu option is active during report preview only.
- **Save** - saves a report.
- **Save as** - saves a report at the path that you specify.
- **Exit** - closes the End User Designer.

### Edit menu

- **Undo** - undoes your last action and enables the **Redo** button.
- **Redo** - redoes an undone action.
- **Cut** - removes the selected item(s) and places them on the clipboard.
- **Copy** - copies the selected item(s) to the clipboard.
- **Paste** - add the last item on the clipboard to the design surface.

### Report menu

- **Convert to CPL Report** - (for an FPL page report) converts an **FPL page report** to a **CPL page report**.
- **Convert to FPL Report** - (for a CPL page report) converts a **CPL page report** to an **FPL page report**.
- **Report Parameters** - opens the **Report Parameters** dialog 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.
- **Set Master Report** - (for a CPL page report) converts a CPL report to a **Master Report**.
- **Generate Excel Template** - (for a CPL page report) creates an Excel template of the report that you or the end user can customize for use with Excel exports to control the way the exported reports appear.
- **View** - opens the Designer, Script or Preview tab. See **Designer Tabs** for more details.
- **Page Header** - (for a CPL page report) toggles the report Page Header on or off.
- **Page Footer** - (for a CPL page report) toggles the report Page Footer on or off.

 **Note:** Except for **View** option, all other options in Report menu are disable for a section report.

For more information on the Designer features, please see **ActiveReports Designer**.

### Project Details

#### ExportForm

This is the form with the **Export** dialog for page 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 (Excel, Html, Tiff, Pdf, Rtf, Text) and to browse for the file location in local folders where the report is exported.

#### ExportForm controls

<b>Control</b>	<b>Name</b>	<b>Description</b>
ComboBox	cmbExportFormat	The <b>Export Format</b> combo box that allows to select options (Excel, Html, Tiff, Pdf, Rtf, Text) for the report export type.
Button	btnOK	The <b>OK</b> button in the lower part of the ExportForm.
Button	btnCancel	The <b>Cancel</b> button in the lower part of the ExportForm.
PropertyGrid	exportpropertyGrid	Provides interface for export options of each export type.
SaveFileDialog	exportSaveFileDialog	The <b>Save File</b> dialog that allows to specify the file name for saving an exported report file.
Label	lblExport	The <b>Export</b> label in the header of the ExportForm.
Label	lblExportFormat	The <b>Export Format</b> label of the Export Format combo box.
Label	lblExportoptions	The <b>Export Options</b> label of the Export property grid.
Label	lblSelectExporttxt	The <b>Select Export</b> text that describes the purpose of the ExportForm.
SplitContainer	splitContainer1	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	splitContainer2	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.

### End User Designer controls

<b>Control</b>	<b>Name</b>	<b>Description</b>
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.
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.
SplitContainer	bodyContainer	Represents a movable bar that divides the display area of the Viewer 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 toolbar.
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.
SplitContainer	MainContainer	Represents a movable bar that divides the display area of the Designer into two resizable panels.
ToolStripContainer	toolStripContainer1	Provides a central panel on top of the Designer to hold the ToolStrip element with the menu items.

Right-click the EndUserDesigner form in the Solution Explorer and select **View Code** to see the code implementation for the End User Designer.

## Professional Web Sample

The Professional Web Sample describes the standard ActiveReports Developer web features as well as other features available in the Professional Edition only, such as HTTP handlers, Flash Viewer options, 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

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Professional\VB.NET\ActiveReports7WebPro
C#
```

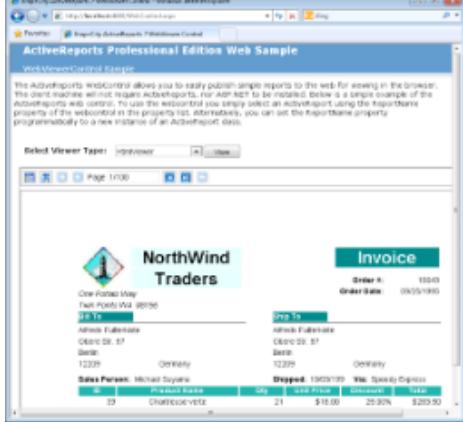
```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Professional\C#\ActiveReports7WebPro
```

#### Runtime Features

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.

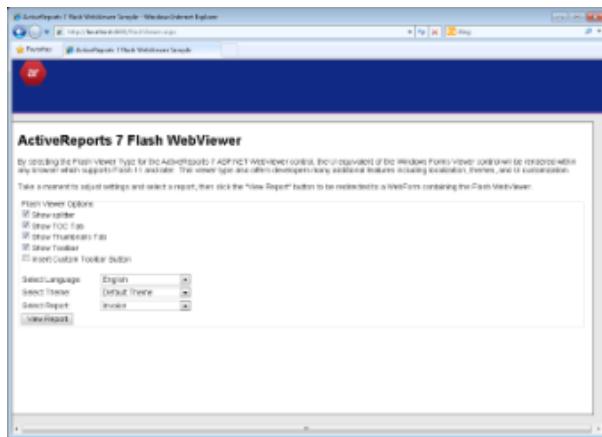
#### WebControl for ASP.NET

This link opens the WebControl.aspx page that displays the Invoice report and allows you to select any of the four Viewer types - HTMLViewer, FlashViewer, PDF Reader and RawHTML.



#### Flash Viewer Control

This link opens the FlashViewerIntro.aspx page where you can adjust settings, select a report and then click the "View Report" button to be redirected to a WebForm with the Flash Viewer.



## HTTPHandlers

This link opens the HttpHandlers.aspx page with the http handler examples.

**ActiveReports RPX and RDLC HttpHandler**

The RPX and RDLC HttpHandler processes and outputs reports that default to layout files ending in the .rpxml extension. When the report is requested by a client, the developer is ending with the .rpxml extension, the RDLC HttpHandler will run, and return the report's output in a format of your choice.

For example:

- [NewLayout.rpx?ReportName=rp1](#)
- [NewLayout.rpx?ReportName=rp2](#)
- [ReportName.rdl?LayoutName=rp1](#)

You can also pass parameters to your reports:

- [NewLayout.rpx?Country=USA](#)
- [NewLayout.rpx?CustomerID=1111&OrderDate=01-1-1994](#)

**ActiveReports Compiled Report HttpHandler**

This compiled Report HttpHandler enables easy distribution of ActiveReports that use compiled JETL source code. If a report requires the raw power, flexibility and efficiency of compiled code, then the Compiled Report HttpHandler is a natural choice. Compiled reports are exposed as a .NET class in a DLL assembly. All you need to do is add the report to the Assembly's resources, and then load the report from the assembly, run it, and return the output in a format of your choice.

The syntax for the HttpHandler is:

<http://www.gnostice.com/ActiveReports/ReportServer/ActiveReport>

For example:

- [ActiveReports/ReportServer/ActiveReport?ReportName=rp1.rpt](#)
- [ActiveReports/ReportServer/ActiveReport?ReportName=rp1.rpt&Country=USA](#)

You can also pass parameters to your reports:

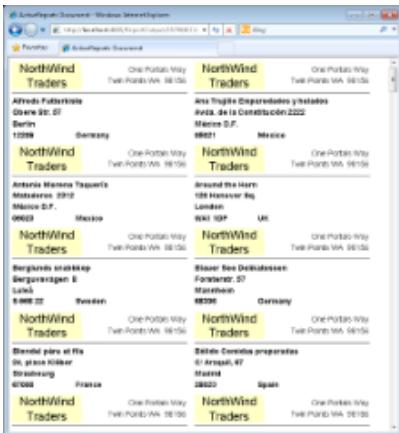
## Custom Exporting PDF Example

This link opens the Invoice report in the PDF Reader by exporting it to memory stream and then outputting it in the browser.

Product Name	Qty	Unit Price	Discount	Total
Chamapgne vert	21	\$16.00	25.00%	\$383.00
Spicewald	2	\$12.00	25.00%	\$18.00
Rössle Sauerkraut	15	\$45.00	25.00%	\$813.00
			<b>Sub Total</b>	<b>\$814.50</b>
			<b>Freight</b>	<b>\$88.38</b>
			<b>Total</b>	<b>\$902.88</b>

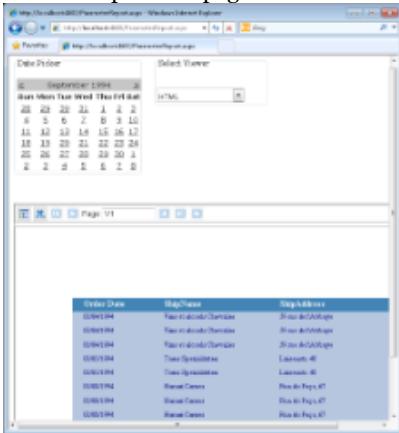
## Custom Exporting HTML Example

This link opens the NWindLabels report. This report is outputted to the ReportOutput folder by using the MyCustomHtmlOutputter class and the exported HTML is displayed in the browser.



## 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**. A copy is located at **[User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data\Nwind.mdb**. If you are unable to access the data files in the sample, create the Data folder and place all the data files to this folder, or change the reference path to the data files within the folder according to your environment.

## Project Details

### 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.

### images

The **images** folder contains the logo.png. This image file is used in the header of the FlashViewerIntro.aspx page.

### RdlxReports

The RdlxReports folder contains the **PurchaseReport** report. This report is used to demonstrate how the RPX and RDLX HttpHandlers function. For more details, see **HttpHandlers.aspx** below.

### RpxReports

The RpxReports folder contains the following reports - **Invoice**, **InvoiceFiltered**, **NwindLabels**, **NwindLabelsFiltered** and **Params**.

The Invoice.rpx report is used to demonstrate the Web Viewer 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.

All reports from this folder are used to demonstrate the Flash Viewer options. You can select one of these reports from the list on the FlashViewer.aspx page.

## Themes

The Themes folder contains themes to use on the Flash Viewer. Following themes can be used for the Flash Viewer.

- FluorescentBlue.swf
- Office.swf
- OliveGreen.swf
- Orange.swf
- VistaAero.swf
- WindowsClassic.swf
- XP.swf

## ActiveReports.ReportService.asmx

The report web service is required for proper functioning of the Web Viewer. The ActiveReports.ReportService.asmx is added automatically to the project when you place the WebViewer control on the form. You can also add this service by using **Add New Item** in the Visual Studio 2010 **Project** menu.

For the information on how to use the WebViewer control, see **Getting Started with the Web Viewer**.

## CustomExportHtml.aspx

This Web form is displayed by clicking the **Custom Exporting HTML Example** option on the Default.aspx page.

In CustomExportHtml.aspx, the NWindLabels report is outputted to the ReportOutput folder using the CustomHtmlOutput class and the exported HTML is displayed in the browser.

 **Note:** This sample requires write permissions to the ReportOutput folder that is located in the web samples directory.

## CustomExportPdf.aspx

This Web form is displayed by clicking the **Custom Exporting PDF Example** option on the Default.aspx page. In CustomExportPdf.aspx, the Invoice 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.

## CustomHtmlOutput class

This class is used for exporting a report to the HTML format. The CustomHtmlOutput class implements the required IOutputHtml in the HTML export and saves the output results to a file with a unique name.

## 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)
- **FlashViewer Control** (FlashViewerIntro.aspx)
- **HTTPHandlers** (HttpHandlers.aspx)
- **Custom Exporting PDF Example** (Invoice report)
- **Custom Exporting HTML Example** (NWindLabels report)
- **Parameterized Report Example** (ParameterReport.aspx)

## FlashViewer.aspx

This is a web form with the Web Viewer that is displayed after you click the **View Report** button on the ActiveReports7 Flash WebViewer Sample.

In the Properties window, notice that the **ViewerType** property is set to **FlashViewer**, and the **Height** and **Width** properties are set to **100%**. (This ensures that the viewer resizes to fill the browser window.)

For information on the Flash Viewer, see **Using the Flash Viewer**.

## FlashViewerIntro.aspx

This web form displays the introductory text for the ActiveReports 7 Flash WebViewer Sample. On this page, you can adjust the Flash Viewer settings, select a language, a theme and a report for the Flash Viewer from the drop-down lists. The report is opened in the Flash

Viewer by clicking the **View Report** button that you see on this page.

This sample uses the reports from the **RpxReports** folder of the Sample project.

Right-click the file and select **View Code** to see the code used to populate the Themes drop-down list and to redirect to the FlashViewer form.

### **Grapecity.ActiveReports.Flash.v7.Resources.swf**

This file is used for localization and is necessary only if you want to use a language resource that is different from the default one. The default locale is U.S. English (en\_US).

This file is located in the ...\\ComponentOne\\ActiveReports Developer 7\\Deployment\\Flash folder.

### **Grapecity.ActiveReports.Flash.v7.swf**

This file is required for using the Flash Viewer and is located in the ...\\ComponentOne\\ActiveReports Developer 7\\Deployment\\Flash folder.

### **HttpHandlers.aspx**

ActiveReports Developer 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 Developer 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 Developer 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 Developer 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 ActiveReport Developer 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 6.x** and **Configure HttpHandlers in IIS 7.x**. The required mapping for each feature has been listed below.

WebViewer control

ActiveReports 7 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)

ActiveReport 7 RPX 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 runtime. 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 - **HTML, Flash, 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 Developer to process reports on the Web.

Note that you need to manually update version information here when you update your version of ActiveReports Developer.

### **WebControl.aspx**

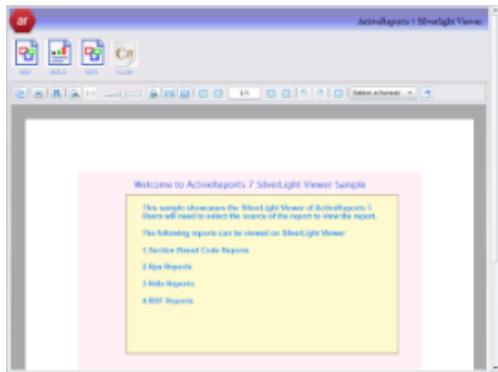
This page is opened by clicking **WebControl for ASP.NET** on the Default.aspx page. It displays the Web Viewer control with the Invoice report.

Using the WebViewer control, you can display the Invoice report on the WebViewerControl Sample page in the following web viewer types - **HTMLViewer, FlashViewer, AcrobatReader, RawHtml**.

The Invoice report is specified in the **ReportName** property of the WebViewer control on the **Design** page of WebControl.aspx.

## Silverlight Viewer Sample

The Silverlight Viewer sample demonstrates the Silverlight Viewer with its various options of loading RPX, RDLX and RDF reports. This Sample is part of the ActiveReports Developer Professional Edition.



### Sample Location

#### Visual Basic.NET

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Professional\VB.NET\SilverLightViewer
C#
```

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Professional\C#\SilverLightViewer
```

#### Runtime Features

When you run the project, the MainPage.xaml page with the Silverlight Viewer displaying the MainReport.rpx appears in your browser, and you will see a number of report options that the Silverlight Viewer will display.

- RDF
- RDLX
- RPX
- Code

### Project Details

#### Images

The Images folder contains images that are used in the header of the ActiveReports7 Silverlight Viewer, including the images for the buttons to load and display an RDF, an RDLX, an RPX and a Code report in the Silverlight Viewer.

#### MainPage.xaml

The user control that contains the Silverlight Viewer. The code behind this file, MainPage.xaml.vb (or .cs), handles the customization of the Silverlight Viewer application.

It contains code that handles the display of RDF, RDLX, RPX, and Code reports in the ActiveReports7 Silverlight Viewer.

#### SilverlightViewer.Web Project

#### Reports

The Reports folder that contains the RDF, RDLX and RPX files of reports used in the sample. You can alter the project and add your own reports to this folder.

#### RDF

The **EmployeeSales.rdf** report that is displayed in the Silverlight Viewer by clicking the **RDF** button in the header of the Viewer.

The report displays the **Sales by Employee** chart and the list of employee names in the alphabetical order along with their total sales.

#### RDLX

The **SalesReport.rpx** is displayed in the Silverlight Viewer by clicking the **RDLX** button in the header of the Viewer. This report

presents the **Sales over time** line chart along with the list of chart data.

The report contains two parameters, **StartDate** and **EndDate**. The report also has one TextBox with the **Label** property set to the expression, thus you can see the Document map of the report in the sidebar pane.

## RPX

The **Invoice1.rpx** and **MainReport.rpx** reports. The Invoice1.rpx report is displayed in the Silverlight Viewer by clicking the **RPX** button in the header of the Viewer. The report displays the invoice form and the invoice details.

The **MainReport.rpx** report is the default report to be displayed in the Viewer when you run the sample project.

## ActiveReports.ReportService.asmx

The report web service required for the proper functioning of the Silverlight Viewer. The ActiveReports.ReportService.asmx is added automatically to the project when you place the Viewer control on the form. You can also add this service by using **Add New Item** in the Visual Studio 2010 **Project** menu.

For the information on how to use the Silverlight Viewer, see **Using the Silverlight Viewer**.

## CodeReport

The **Invoice** report that is displayed in the Silverlight Viewer by clicking the **Code** button in the header of the Viewer.

### Invoice Report

The Invoice report uses a PageHeader, a GroupHeader, a Detail section, a GroupFooter and a PageFooter section.

#### PageHeader

This section contains Label, Textbox, Shape and Line controls to create the report page header.

#### customerGroupHeader

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 **Grouping Data in Section Reports**.

This section also contains the Line, Shape, CrossSection controls, a number of Label controls, and two bound TextBox controls. The TextBoxes are bound using the **DataField** property in the Properties window.

In the lower part, the section contains 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 section.

#### customerGroupFooter

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. The subtotalTextBox control uses the following properties to summarize the detail data: **SummaryRunning**, **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 ReportInfo control. For more information about report sections and the order in which they print, see **Section Report Structure** and **Section Report Events**.

#### Default.html

This is the html page that hosts the Silverlight Viewer in the browser.

#### Report.aspx

This report contains the code to run the CodeReport, generate it in the rdf format and pass it to the Silverlight Viewer for display. This report is processed by the code of MainPage.xaml when you click the **Code** button in the Silverlight Viewer header.

## Web.config

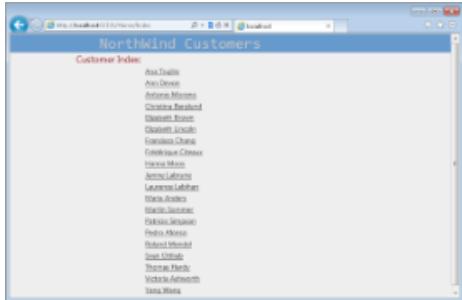
The configuration file that contains the httpHandlers that allow ActiveReports Developer to process reports on the Web.

Note that you need to manually update version information here when you update your version of ActiveReports Developer.

## ActiveReports 7 with MVC Sample

The ActiveReports7 with MVC sample demonstrates an MVC web application with the ActiveReports Web Viewer. This Sample is part of the ActiveReports Developer Professional Edition.

 **Note:** Before running this sample, you must first install the ASP.NET MVC 3 framework on your machine.



### Sample Location

#### Visual Basic.NET

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Professional\VB.NET\ActiveReports7WithMVC\code>
C#
```

```
<User Folder>\ComponentOne Samples\ActiveReports Developer 7\Professional\C#\ActiveReports7WithMVC
```

#### Runtime Features

When you run the project, the page with the **Customer Index** view appears. On this page, you can click a customer in the list to open a new page with the **Customer Details** information. At the bottom of the page, there are two links - **Freight Summary (section report)** and **Order Summary (page report)**. Clicking the **Freight Summary (section report)** link opens a page with the **OrderReport** report. Clicking the **Order Summary (page report)** link opens a page with the **OrderDetailsReport.rdlx** report.

#### Project Details

##### Controllers

This folder contains the **HomeController** that handles the user interaction and returns the main Index view, the CustomerDetails view and the views with Freight Summary (section report) and Order Summary (page report).

##### DBContext

This folder contains the **NWindData** class that gets the Customer, Orders, Order Details tables of the NWIND.mdb and populates the data into the components of this sample project.

##### Models

This folder contains the following classes.

**Customer** - a container for the rows in the Customers table of NWind.mdb.

**Order** - a container for the rows in the Orders table of NWind.mdb.

**OrderDetails** - a container for the rows in the Order Details table of NWind.mdb.

**ReportDescriptor** - the class representation of the ReportType (section or page layout) and the CustomerID.

**Repository** - a container for data required by the application.

##### Reports

**OrderDetailsReport.rdlx**: This report uses the **object provider** to connect to data at runtime. It uses the **Table** data region to display order details and the Plain Line chart to display sales by order id. The Table data is grouped by **OrderID**. Some TextBox controls of the Table use the **Sum function** to display the total price information for each order.

This is the report that opens in a separate view when you click the **Order Summary (page report)** link on the Customer Details page.

**OrderReport**: This report binds to data at runtime. The report uses the Textbox and Label controls to display the order details and the Bar chart to display the freight amounts by ship date.

This is the report that opens in a separate view when you click the **Freight Summary (section report)** link on the Customer Details page.

#### Views

##### Home folder

This folder contains the following views.

**\_Layout.cshtml**. The default layout file.

**\_ViewStart.cshtml.** The default view start file.

**Details.cshtml.** The Customer Details view file.

**Index.cshtml.** The Index view file.

**StartPage.cshtml.** The template for the Index view.

**Viewer.cshtml.** The template for the section report and page report views.

**WebViewer.ascx.** The form that contains the ActiveReports WebViewer control.

## Web.config

The configuration file that contains the httpHandlers that allow ActiveReports Developer to process view pages on the Web.

## ActiveReports.ReportService.asmx

The report web service required for the proper functioning of the Web Viewer. The ActiveReports.ReportService.asmx is added automatically to the project when you place the Viewer control on the form. You can also add this service by using **Add New Item** in the Visual Studio 2010 **Project** menu.

## Global.asax

The default class that sets global URL routing values for this web application.

## Web.config

The configuration file that contains the httpHandlers that allow ActiveReports Developer to process this web application.

Note that you need to manually update version information here when you update your version of ActiveReports Developer.

## 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 and Section Reports in different scenarios. The walkthroughs progress from basic through advanced for Standard and Professional Editions of ActiveReports.

## Page Report Walkthroughs

Learn the dynamic features in page reports, through easy to implement scenarios for Fixed Page Layout (FPL) and Continuous Page Layout (CPL) report types.

### General Page Report Walkthroughs

#### Custom Data Provider

#### Reports with XML Data

#### BandedList Reports

#### Matrix Reports

#### Master Detail Reports

#### Expressions in Reports

#### Recursive Hierarchy Reports

#### Reports with Parameterized Queries

#### Reports with Custom Code

#### Reports with Stored Procedures

#### Charts in a Page Report

#### Interactive Reports

#### Custom Web Exporting

## Fixed Page Layout (FPL) Walkthroughs

[Single Layout Reports](#)

[Overflow Data in Multiple Pages](#)

[Overflow Data in a Single Page](#)

[Collate Multiple Copies of a Report](#)

## Continuous Page Layout (CPL) Walkthroughs

[Subreport in a CPL Report](#)

[Columnar Layout Reports \(CPL\)](#)

# Section Report Walkthroughs

Use the walkthroughs under this section to understand key features of a section report through simple scenarios.

## Basic Data Bound Reports

[Basic XML-Based Reports \(RPX\)](#)

[Address Labels](#)

[Columnar Reports](#)

[Overlaying Reports \(Letterhead\)](#)

[Chart Walkthroughs](#)

[Custom HTML Outputer](#)

[Custom Web Exporting \(Std Edition\)](#)

[Group On Unbound Fields](#)

[Subreport Walkthroughs](#)

[Mail Merge with RichText](#)

[Run Time or Ad Hoc Reporting](#)

[Layout Files with Embedded Script](#)

[Basic Spreadsheet with SpreadBuilder](#)

[Web Services \(Standard Edition\)](#)

## Professional Edition Walkthroughs

[Creating a Basic End User Report Designer \(Pro Edition\)](#)

[Customizing the Flash Viewer UI](#)

[Customizing the HTML Viewer UI](#)

# General Walkthroughs

[WPF Viewer](#)

## Page Report Walkthroughs

Page Report walkthroughs comprise of scenarios to introduce the key features of Fixed Page Layout and Continuous Page Layout reports.

## General Page Report Walkthroughs

**BandedList Reports**

**Matrix Reports**

**Reports with XML Data**

**Master Detail Reports**

**Reports with Parameterized Queries**

**Reports with Custom Code**

**Matrix Reports**

**Expressions in Reports**

**Recursive Hierarchy Reports**

**Reports with Stored Procedures**

**Custom Data Provider**

**Charts**

**Interactive Reports**

**Custom Web Exporting**

## Fixed Page Layout (FPL) Walkthroughs

**Single Layout Reports**

**Overflow Data in Multiple Pages**

**Overflow Data in a Single Page**

**Collate Multiple Copies of a Report**

## Continuous Page Layout (CPL) Walkthroughs

**Subreport in a CPL Report**

**Columnar Layout Reports (CPL)**

## Single Layout Reports

A fixed page layout, 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 runtime. This walkthrough illustrates how to create a report that contains a single layout and preview it.

This walkthrough is split into the following activities:

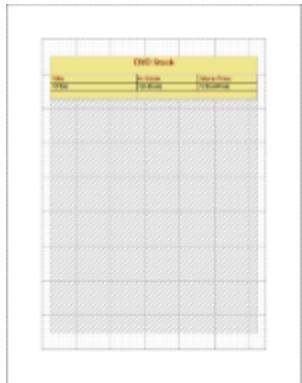
- Adding an ActiveReport to a Visual Studio project
- Connecting to a the data source
- Adding a dataset
- Creating a layout
- Viewing the report

 **Note:** This walkthrough uses the **DVDStock** table from the Reels database. By default, in ActiveReports, the Reels.mdb file is located at [User Documents folder]\ComponentOne Samples\ActiveReports Developer

7\Samples\Data\Reels.mdb.

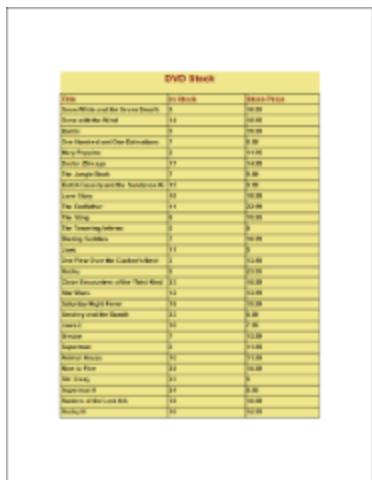
When you complete this walkthrough you get a layout that looks similar to the following at design time and at runtime.

## Design Time Layout



Notice that the runtime report layout below is similar to the one you see at design time except for the data which you see at runtime or when you preview the report.

## Runtime Layout



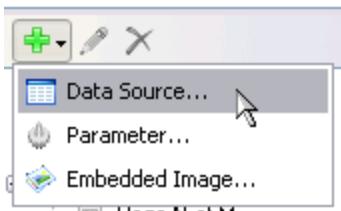
### To add an ActiveReport to the Visual Studio project

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 7 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 **Adding an ActiveReport to a Project** 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 DVList. 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

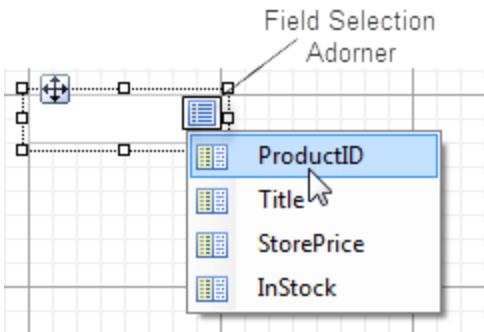
1. In the Visual Studio toolbox, go to the **ActiveReports 7 Page Report** tab and drag a **TextBox** control onto the design surface.
2. Select the TextBox control and go to the Properties window to set the following properties.

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

- In the Table data region, place your mouse over the cells of the table details row to display the field selection adorner.



- 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.

- 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

- 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 **Using the Viewer** for further information.

## Overflow Data in Multiple Pages

In a fixed page layout 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 **OverflowPlaceHolder** control.

This walkthrough is split into the following activities:

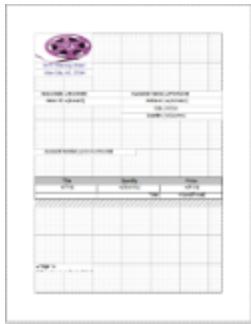
- Adding an ActiveReport to a Visual Studio project
- Connecting to a the data source
- Adding a dataset
- Creating a layout for the first page
- Creating a layout for subsequent pages
- Viewing the Report

 **Note:** This walkthrough uses the **CustomerOrders** table from the Reels database. By default, in ActiveReports, the Reels.mdb file is located at [User Documents folder]\ComponentOne Samples\ActiveReports Developer \Data\Reels.mdb.

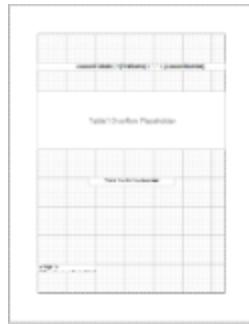
When you complete this walkthrough you get a layout that looks similar to the following at design time and at run time.

### Design Time Layout

**Page 1**



**Page 2**

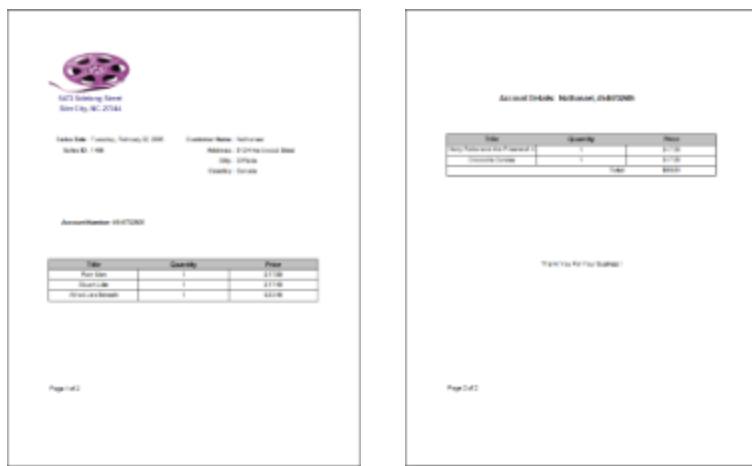


Notice that the runtime report layout below is similar to the one you see at design time except for the data which is added to the report at runtime or when you preview it.

### Runtime Layout

**Page 1**

**Page 2**



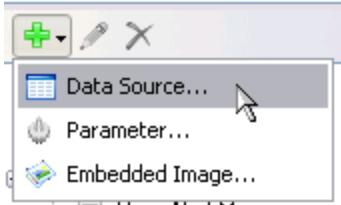
## To add an ActiveReport to the Visual Studio project

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 7 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 **Adding an ActiveReport to a Project** 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 **Grouping in a FixedPage** 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 7 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, 2.5in 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, 2in  
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

- 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

- 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.

- 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

- 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: <code>=Sum(Fields!Price.Value)</code>

- 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

- Click the **New** tab to add a new page to the report layout. This page is named **Page 2** by default.
- From the Visual Studio toolbox, drag the following controls from the **ActiveReports 7 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: <code>=Fields!FirstName.Value + ", " + Fields!AccountNumber.Value</code>
OverflowPlaceHolder	Location: 0in, 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 **Using the Viewer** to display report in the Viewer at runtime.

## Overflow Data in a Single Page

In a fixed page layout, 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. By default, in ActiveReports, the Reels.mdb file is located at [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data\Reels.mdb.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at runtime.

## Design Time Layout



Notice that the runtime report layout below is similar to the one you see at design time except for the data which you see at runtime or when you preview the report.

## Runtime Layout



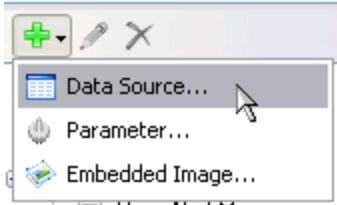
### To add an ActiveReport to the Visual Studio project

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 7 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 [Adding an ActiveReport to a Project](#) 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 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 7 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 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 OverflowName: OverflowPlaceholder2
OverflowPlaceholder2	Location: 0.125in, 5in OverflowName: OverflowPlaceholder3
OverflowPlaceholder3	Location: 3.375in, 5in

 **Note:** Set this property after you add the OverflowPlaceholder2 to the design surface.

 **Note:** Set this property after you add the OverflowPlaceholder3 to the design surface.

OverflowPlaceholder3 Location: 3.375in, 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 **Using the Viewer** to display report in the Viewer at runtime.

## Collate Multiple Copies of a Report

In a fixed page layout 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.

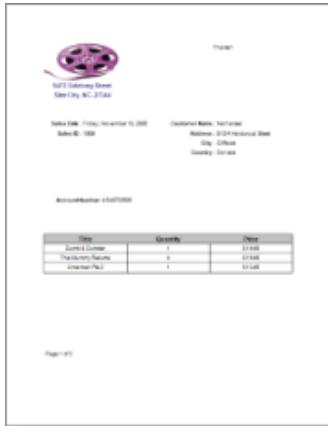
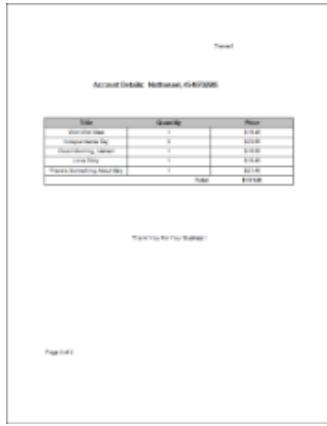
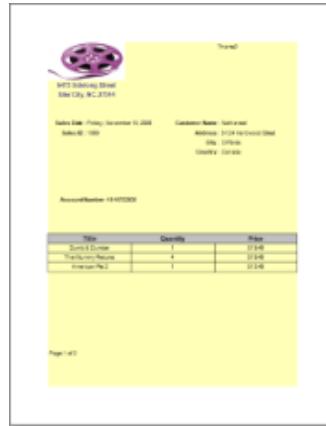
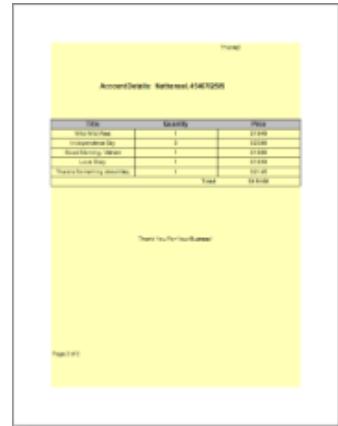
This walkthrough is split into the following activities:

- 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. By default, in ActiveReports, the Reels.mdb file is located at [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data\Reels.mdb.

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 fixed page layout, 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 ( <b>Colors</b> tab)	Default (white)	To set the background color for the theme.
Name ( <b>Constants</b> tab)	Constant1	To define a name for the constant expression to be used within in a theme.
Value ( <b>Constants</b> tab)	Theme1	To associate a value to be used as a constant in the expression.
Description ( <b>Constants</b> 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:

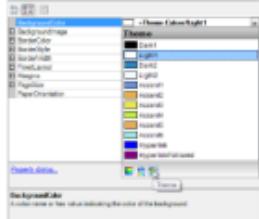
#### Theme 2

<b>Property Name</b>	<b>Property Value</b>	<b>Description</b>
Text/Background - Light1 ( <b>Colors</b> tab)	Yellow	To set the background color for the theme.
Name ( <b>Constants</b> tab)	Constant1	To define a name for the constant expression to be used within in a theme.
Value ( <b>Constants</b> tab)	Theme2	To associate a value to be used as a constant in the expression.
Description ( <b>Constants</b> 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 7 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.

<b>Property Name</b>	<b>Property Value</b>
----------------------	-----------------------

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 **Expression Editor** option that appears when you click the ellipses (...) button adjacent to the Value property.

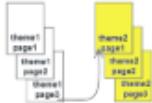
9. From the Visual Studio toolbox, drag the **Textbox** control and drop it onto the Page 2 design surface. Set the properties specified in step 8 above in the properties window.

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** (**'CollateBy Property' in the on-line documentation**) property.

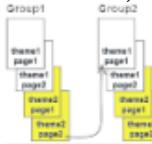
1. In the Designer, click the gray area around the report page to select the report.
2. In the Properties Window, go to the CollateBy property and select the **Simple** mode. The report will be rendered in the following way - the report renders all pages with theme 1, then all pages with theme 2.



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 **Using the Viewer** to display report in the Viewer at runtime.

## Subreport in a CPL Report

You can create a CPL report that hosts a subreport. This walkthrough illustrates how to create a 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. By default, in ActiveReports Developer, the Reels.mdb file is located at [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data\Reels.mdb.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at runtime.

## Design Time Layout



## Runtime Layout



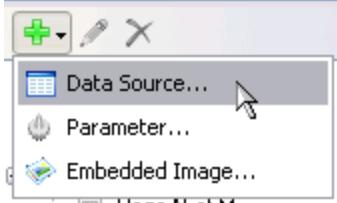
### To add a report for the subreport

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 7 Page Report** and in the Name field, rename the file as **Sales.rdlx**.
4. Click the **Add** button to open a new fixed page report in the **designer**.
5. From the **Report** menu, select **Convert to CPL Report**.

See **Adding an ActiveReport to a Project** 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 property 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
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



 **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
------------	-------------	---------------

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.

- In the **Report Explorer**, from the **EmployeeSales** dataset, drag the following fields onto the group footer row of the table.

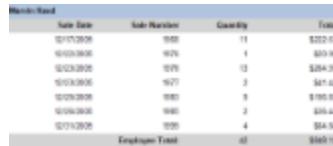
<b>Data Field</b>	<b>Column Name</b>	<b>Property Name</b>
Quantity	TableColumn3	Value: =Sum(Fields!Quantity.Value)
Total	TableColumn4	Format: Currency Value: =Sum(Fields!Total.Value)

- Enter the following text into the indicated cell in the group footer row of the table.

<b>Text</b>	<b>Column Name</b>	<b>Property Name</b>
Employee Total:	TableColumn2	FontWeight: Bold TextAlign: Right

- Using the handle to the left of the group footer row, select the row and in the **BackgroundColor** property select **LightGray**.
- 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 runtime.

### Design Time Layout      Runtime Layout

Design Time Layout					Runtime Layout				
									

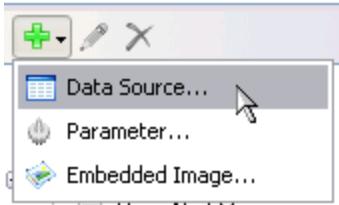
- From the **File** menu, select **Save** and save this file. This report functions as the subreport you use in the main report.

### To create the main report

- From the Visual Studio **Project** menu, select **Add New Item**.
- In the Add New Item dialog that appears, select **ActiveReports 7 Page Report** and in the Name field, rename the file as **Employees.rdlx**.
- Click the **Add** button to open a new fixed page report in the **designer**.
- From the **Report** menu, select **Convert to CPL Report**.

### To connect the main report to a data source

- 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.



- 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, 0.375in
Size	6.5in, 2.875in

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	oin, oin
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 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	Beige
DataSetName	EmployeeInfo
Location	0.125in, 0.3in
Size	6.25in, 2.5in

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	oin, oin
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 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	White
DataSetName	EmployeeInfo
Location	oin, 0.25in
Size	6.125in, 1.875in

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, oin 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

- From the toolbox, drag five text boxes onto the List and set the following properties:

<b>TextBox Name</b>	<b>Value Property</b>	<b>Property Name</b>
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

- 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:

<b>Property Name</b>	<b>Property Value</b>
Location	0.125in, 0.875in
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	5.875in, 0.875in
Visibility: Hidden	True (hides the subreport initially)
Visibility: ToggleItem	Sales Record text box added in the previous 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 **Using the Viewer** to display report in the Viewer at runtime.

 **Note:** Click the + to the left of **Sales Record** to see the subreport.

## Columnar Layout Reports (CPL)

In CPL reports, you can create a columnar report layout by using the **Columns** property of the report. This walkthrough illustrates how to create a CPL 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. By default, in ActiveReports, the Reels.mdb file is located in the [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data folder.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at runtime.

## Design Time Layout



## Runtime Layout



## To add an ActiveReport to the Visual Studio project

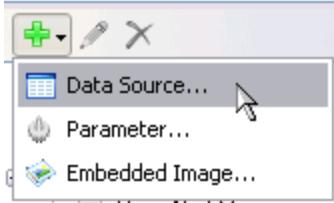
1. Create a new Visual Studio project.

2. From the Project menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 7 Page Report** and in the Name field, rename the file as rptCPLColumnLayout.
4. Click the **Add** button to open a new fixed page report in the **designer**.
5. From the Report menu, select **Convert to CPL Report**.

See **Adding an ActiveReport to a Project** 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 7 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 7 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 **Using the Viewer** for further information.

If you want to create a similar layout in FPL reports, see **Overflow Data in a Single Page (on-line documentation)**.

## BandedList Reports

You can create a freeform report with groups using the **BandedList** 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. By default, in ActiveReports Developer, the Reels.mdb file is located in the [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data folder.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at runtime.

## Design Time Layout



## Runtime Layout



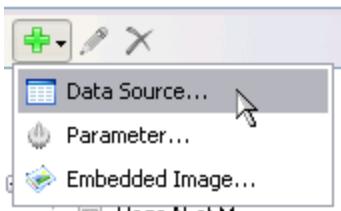
### To add an ActiveReport to the Visual Studio project

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 7 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 **Adding an ActiveReport to a Project** 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 adorer (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 adorer (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 adorer (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 an FPL 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 **Using the Viewer** for further information.

## Matrix Reports

Use nested grouping and subtotals in the **Matrix** data region, to replicate a drilldown report. This walkthrough illustrates a step by step overview of a Matrix report through a simple example.

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 with calculated fields
- Creating a layout for the report
- Adding nested grouping and drill-down to the matrix
- Adding subtotals to the matrix
- Enhancing the appearance of the report
- Viewing the report

 **Note:** This walkthrough uses the **Sale** table from the Reels database. By default, in ActiveReports Developer, the Reels.mdb file is located in the [User Documents folder]\ComponentOne Samples\ActiveReports Developer \Data folder.

When you complete this walkthrough you get a layout looks similar to the following at design time and at runtime.

## Design Time Layout



## Runtime Layout

Sales by Store							Grand Total
	2004	Q1	Q2	Q3	Q4	Grand Total	
Store Number 4880	\$8,589.10	\$1,980.00	\$3,750.00	\$3,980.00	\$2,980.00	\$16,230.00	\$20,707.30
Store Number 4881	\$6,160.40	\$2,012.17	\$3,090.17	\$3,090.17	\$2,968.20	\$12,702.00	\$16,809.32
Store Number 4882	\$4,858.38	\$1,034.54	\$4,080.05	\$3,380.03	\$6,181.20	\$15,718.00	\$25,415.38
Store Number 4883	\$8,795.22	\$1,913.77	\$3,121.16	\$2,721.26	\$9,436.11	\$17,808.12	\$22,396.38
Store Number 4884	\$8,166.52	\$2,250.00	\$2,917.00	\$2,286.00	\$2,943.07	\$16,408.00	\$19,813.52
Store Number 4885	\$8,565.00	\$1,465.00	\$2,025.00	\$3,764.50	\$4,372.00	\$17,219.50	\$19,775.52
Store Number 4886	\$8,414.00	\$3,371.74	\$2,085.00	\$2,989.14	\$3,941.43	\$15,348.00	\$20,461.41

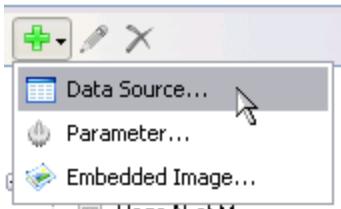
### To add an ActiveReport to the Visual Studio project

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 7 Page Report** and in the Name field, rename the file as rptMatrix.
4. Click the **Add** button to open a new fixed page report in the **designer**.

See **Adding an ActiveReport to a Project** 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 Sale. 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, SaleDate, SalesAmount FROM Sale ORDER BY Store, SaleDate
```

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 of this dialog, enter three new calculated fields with the values in the following table.

Field Name	Value
Month	=Fields!SaleDate.Value.Month
Quarter	=Choose(1 + ((Fields!Month.Value - 1)\3), "Q1", "Q2", "Q3", "Q4")
Year	=Fields!SaleDate.Value.Year

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. Click the gray area below the design surface to select the report, go to the Properties Window, expand the **PageSize** property and set the **Width** to 10in and **Height** 11in.
2. From the toolbox, drag a **Matrix** data region onto the body of the report in the top left corner. The matrix initially has four textbox cells.

Property Name	Property Value
Location	0in, 0.5in
Size	3in, 1in
FixedSize (only for FPL reports)	8in, 7in

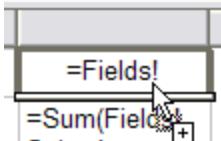
3. In the **Report Explorer** from the **Sale** dataset, drag the **Store** field into the bottom left cell in the matrix. This is the row header, and dragging a field into it automatically adds a Row Group which groups data by store in rows down the page at run time.
4. Go to the **Properties Window** to set the **Value** property to ="Store #" & Fields!Store.Value. This concatenates a string with the field value and makes the Store value more user-friendly at run time.
5. In the Report Explorer from the **Sale** dataset, drag the **Year** field into the top right cell in the matrix. This is the

column header, and automatically groups data in columns to the right at runtime.

6. With the Year cell selected, go to the Properties window to set the  **TextAlign** property to  **Center**.
7. In the Report Explorer from the **Sale** dataset, drag the **SalesAmount** field into the bottom right cell in the matrix. This is the detail data cell, and displays aggregated data for the intersection of each column and row at run time.
8. With the SalesAmount cell selected, in the Properties window set the  **Format** property to Currency and the  **Value** property to `=Sum(Fields!SalesAmount.Value)`.

#### To add nested grouping and drilldown features to the matrix

1. In the **Report Explorer** from the **Sale** dataset, drag the **Quarter** field to the top right Year cell and move the cursor down slightly so that the bar appears at the bottom edge of the cell before dropping it.



This adds a cell below the Year cell and automatically adds a Quarter grouping to the Column Groups for the matrix.

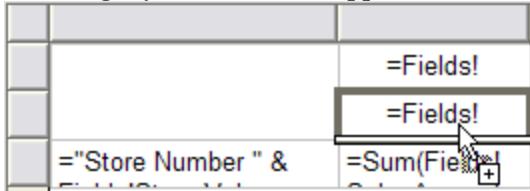
2. With the new Quarter cell selected, go to the Properties window to set the  **TextAlign** property to  **Center**.
3. Select the Matrix data region and under the Properties Window select **Property dialog** command to open the Matrix dialog. See **Properties Window** for details on commands.
4. On the **Column Groups** page, select the **Matrix1\_Quarter** group and go to the **Visibility** tab.
5. Under **Visibility**, change the **Initial visibility** to **Hidden** and select the check box next to **Visibility can be toggled by another report item**.

**Tip:** In order to provide drill-down functionality, we must set visibility on the *group* we want to hide.

6. In the drop down enabled below, enter **TextBox3** (Textbox containing Year values in the Matrix) and click **OK** to close the dialog.

This enables users to click the plus sign next to the Year to view the quarterly details at runtime.

7. In the Report Explorer from the **Sale** dataset, drag the **Month** field to the Quarter cell and move the cursor down slightly so that the bar appears at the bottom edge of the cell before dropping it.



This adds a cell below the Year cell and automatically adds a Month grouping to the Column Groups for the matrix.

8. Select the Matrix data region and under the Properties Window select **Property dialog** command to open the Matrix dialog.
9. On the **Column Groups** page, select the **Matrix1\_Month** group and go to the **Visibility** tab.
10. Under **Visibility**, change the **Initial visibility** to **Hidden** and select the check box next to **Visibility can be toggled by another report item**.
11. In the drop down enabled below, enter **TextBox5** (Textbox containing Quarter values in the Matrix) and click **OK** to close the dialog.

This enables users to click the plus sign next to the quarter to view the monthly details at runtime.

12. Click **OK** to close the dialog.

#### To add subtotals to the matrix

Without subtotals, viewing the report at run time shows data for each year, or if expanded, for each quarter or month. Adding subtotals displays totals for each group in a new column to the right of the group.

- Right-click the Year cell and select **Subtotal**. A new column appears to the right with the text **Total** and a green mark at the top right corner (Clicking the green mark displays the **MatrixSubtotal** properties in the **Properties Window**).
- Select the new Total cell and go the Properties Window to set the following properties:

Property Name	Property Value
TextAlign	Center
Value	Grand Total

- Right-click the **Quarter** cell and select **Subtotal**.
- Select the new Total cell and go to the Properties Window to set the following properties:

Property Name	Property Value
TextAlign	Center
Value	=Fields!Year.Value & " Total"

- Right-click the **Month** cell and select **Subtotal**.
- Select the new Total cell and go to the Properties Window to set the following properties:

Property Name	Property Value
TextAlign	Center
Value	=Fields!Quarter.Value & " Total"

### To enhance the appearance of the report

If you preview the report at this point, you notice that although all of the data displays correctly, it is difficult to identify data when you start drilling down into it. This can be improved with background colors and borders.

- In the matrix data region of your report, select the textbox that contains the month field and go to the **Properties Window** to set the following properties:

Property Name	Property Value
Format	MMMM
Value	=Fields!SaleDate.Value

 **Note:** Changing the textbox value does not affect the grouping value, so the data is still grouped by month.

- Holding down the **Shift** key, select the two textboxes in the top row containing the Year and Grand Total cells and make the following changes in the Properties window:

Property Name	Property Value
BackgroundColor	Gainsboro
BorderStyle	Solid

- Holding down the Shift key, select the two textboxes in the second row containing the expressions `=Fields!Quarter.Value` and `=Fields!Year.Value & " Total"` and make the following changes in the Properties Window:

Property Name	Property Value
BackgroundColor	LightSteelBlue
BorderStyle	Solid

- Holding down the Shift key, select the two textboxes in the third row containing the month and quarterly total

cells and make the following changes in the Properties Window:

Property Name	Property Value
---------------	----------------

BackgroundColor	AliceBlue
BorderStyle	Solid

5. Holding down the Shift key, select the two textboxes in the fourth row containing the store number and detail cells and make the following change in the Properties Window:

Property Name	Property Value
---------------	----------------

BorderStyle	Solid
-------------	-------

6. From the toolbox, drag a **Textbox** control onto the design surface to span the entire width of the report.

 **Tip:** In a CPL report, you can place the Textbox in the PageHeader.

7. Go to the Properties window to set the following properties.

Property Name	Property Value
---------------	----------------

Location	0in, 0.25in
Size	6.5in, 0.25in
TextAlign	Center
FontSize	14pt
Value	Sales by Store

## To view the report

- Click the preview tab to view the report at design time.  
OR
- Open the report in the Viewer. See **Using the Viewer** for further information.

## Reports with XML Data

This walkthrough 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.

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
- Viewing the report

 **Note:** This walkthrough uses the **Factbook** sample database. By default, in ActiveReports Developer, the Factbook.rdsx file is located at [User Documents folder]\ComponentOne Samples\ActiveReports Developer\7\Data\Factbook.rdsx.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at runtime.

## Design Time Layout



## Runtime Layout



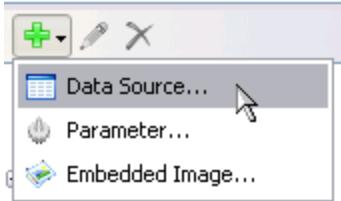
### To add an ActiveReport to the Visual Studio project

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 7 Page Report** and in the Name field, rename the file as ExchangeRates.
4. Click the **Add** button to open a new page report in the **designer**.

See [Adding an ActiveReport to a Project](#) 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, which is located in [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data. 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
Name	Database	@name

		Field
Currency	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.
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
2004	Location: 4.5in, 0.875in Size: 1in, 0.25in
2003	Location: 4.5in, 1.25in Size: 1in, 0.25in
2002	Location: 4.5in, 1.625in Size: 1in, 0.25in
2001	Location: 4.5in, 2in Size: 1in, 0.25in
2000	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**

<b>Property Name</b>	<b>Property Value</b>
Location	0.125in, 0.5in
Size	0.75in, 0.25in
FontWeight	Bold
Value	Currency:

**TextBox2**

<b>Property Name</b>	<b>Property Value</b>
Location	3.375in, 0.875in
Size	1in, 0.25in
TextAlign	Right
Value	2004:

**TextBox3**

<b>Property Name</b>	<b>Property Value</b>
Location	3.375in, 1.25in
Size	1in, 0.25in
TextAlign	Right
Value	2003:

**TextBox4**

<b>Property Name</b>	<b>Property Value</b>
Location	3.375in, 1.625in
Size	1in, 0.25in
TextAlign	Right
Value	2002:

**TextBox5**

<b>Property Name</b>	<b>Property Value</b>
Location	3.375in, 2in
Size	1in, 0.25in
TextAlign	Right
Value	2001:

**TextBox6**

<b>Property Name</b>	<b>Property Value</b>

Location	3.375in, 2.375in
Size	1in, 0.25in
TextAlign	Right
Value	2000:

### To view the report

- Click the preview tab to view the report at design time.
- OR
- Open the report in the Viewer. See **Using the Viewer** for further information.

## 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. By default, in ActiveReports Developer, the Reels.mdb file is located in the [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data folder.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at runtime.

## Design Time Layout



## Runtime Layout

Kommery			
Title	Quantity	Price	Total
Rain-Man	1	\$17.00	\$17.00
Shawshank	1	\$17.49	\$17.49
What Lies Beneath	1	\$23.45	\$23.45
Henry Fonda and the Pleasures of Addiction	1	\$17.99	\$17.99
Desperado	1	\$17.00	\$17.00
Unsane	1	\$16.49	\$16.49
Good Morning, Vietnam	1	\$16.99	\$16.99
Brave & Bold	1	\$16.40	\$16.40
Independence Day	3	\$23.95	\$71.85
There's Something About Mary	1	\$16.95	\$16.95
Wild Wild West	1	\$16.49	\$16.49
American Pie 2	1	\$16.40	\$16.40
The Mummy Returns	4	\$16.45	\$65.80
<b>Total:</b>			

Fright			
Title	Quantity	Price	Total
Troy	1	\$23.49	\$23.49
Mr. & Mrs. Smith	1	\$23.95	\$23.95

### To add an ActiveReport to the Visual Studio project

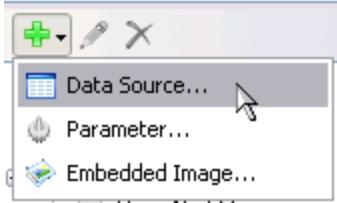
- Create a new Visual Studio project.
- From the **Project** menu, select **Add New Item**.

3. In the Add New Item dialog that appears, select **ActiveReports 7 Page Report** and in the Name field, rename the file as rptMasterDetail.
4. Click the **Add** button to open a new fixed page report in the **designer**.

See **Adding an ActiveReport to a Project** 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 after 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 adorer.
2. Click the adorer 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 adorer 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 adorer 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 adorer to select the **Price** field.
13. In the **Properties window**, set the following properties.

<b>Property Name</b>	<b>Property Value</b>
----------------------	-----------------------

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.

<b>Property Name</b>	<b>Property Value</b>
----------------------	-----------------------

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.

<b>Property Name</b>	<b>Property Value</b>
----------------------	-----------------------

RepeatOnNewPage	True
-----------------	------

- |            |      |
|------------|------|
| FontWeight | Bold |
|------------|------|
- 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.
  - Go to the **Properties window** to set the following properties.

Property Name	Property Value
---------------	----------------

Format	C (uses currency formatting)
FontWeight	Bold

- From the **Report Explorer**, drag the **Total** field into the table footer row in the fourth column. This adds grand totaling to the table.
- Go to the **Properties window** to set the following properties.

Property Name	Property Value
---------------	----------------

Format	C (uses currency formatting)
FontWeight	Bold

- Delete the static label **Total** from the top row.
- Click the gray area below the design surface to give report the focus and from the **Report** menu, select **Page Header**.

 **Note:** FPL reports do not have page header and page footer options. Go to the last step of this section to finish your FPL report.

- From the toolbox, drag the **TextBox** control onto the PageHeader section to span the entire width of the report.
- Go to the **Properties Window** to set the following properties.

Property Name	Property Value
---------------	----------------

TextAlign	Center
FontSize	14pt
Value	Customer Orders

- For an FPL report, in the Report Explorer select the Table data region and 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 **Using the Viewer** for further information.

## 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 walkthrough illustrates the how to use expressions to achieve different effects.

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
- Adding a field expression to a text box to multiply two field values
- Adding an Immediate If expression to show or hide a control

- Adding a Data Visualization expression to display data graphically
- Viewing the report

 **Note:** This walkthrough uses the **MovieProduct** table from the Reels database. By default, in ActiveReports Developer, the Reels.mdb file is located in the [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data folder.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at runtime.

## Design Time Layout



## Runtime 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 7 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 **Adding an ActiveReport to a Project** 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 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.

### To 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 FPL report, in the Report Explorer select the Table control and in the Properties window, set the **FixedSize** property to **6.5in, 7in**.

#### To 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.

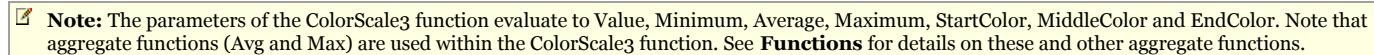
#### To add an Immediate If expression to show or hide a report item

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)`

#### To 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.

#### To view the report

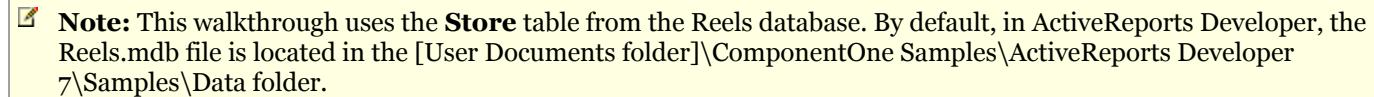
- Click the preview tab to view the report at design time.
- OR
- Open the report in the Viewer. See **Using the Viewer** for further information.

## 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. By default, in ActiveReports Developer, the Reels.mdb file is located in the [User Documents folder]\ComponentOne Samples\ActiveReports Developer \Samples\Data folder.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at runtime.

## Design Time Layout



## Runtime Layout

Store Number 1200			
Title	Last Name and ID	Supervisor ID	Department
Store Manager	Weller 1001	0	Store Management
Store Assistant Manager	Mosberg 1009	4001	Store Management
Store Associate - Systems	Roskelly 1013	4002	Store Inventory Systems
Store Assistant Manager	Brennake 1015	4007	Store Management
Store Associate - Systems	Arie 1016	4008	Store Inventory Systems
Store Shift Supervisor	Adley 1017	4028	Store Management
Store Worker	Morgan 1040	6001	Store Workers
Store Shift Supervisor	Yann 1050	6002	Store Management
Store Worker	Reilly 1067	6003	Store Workers
Store Worker	Fleck 1029	6005	Store Workers
Store Worker	Amesouse 1038	6006	Store Workers
Store Cashier	Margate 1077	6009	Store Cashiers
Store Cashier	Grape 1086	6010	Store Cashiers
Store Cashier	Roxie 1085	6012	Store Cashiers
Store Cashier	Lippsen 1014	6013	Store Cashiers

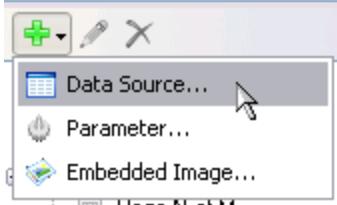
### To add an ActiveReport to the Visual Studio project

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 7 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 [Adding an ActiveReport to a Project](#) 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): 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	oin, oin
DataSetName	Employees
FixedSize (only for FPL 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 detail 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. Right-click the table handle to the left of the header row and select **Insert Row Above**.
13. While holding down the SHIFT key, click each of the cells in the newly added top row.
14. Right-click inside the selected cells and select **Merge Cells** to create a cell that spans the table.
15. Go to the **Properties window** to set the following properties.

Property Name	Property Value
TextAlign	Center
FontSize	14pt
Value	= "Store Number " & Parameters!StoreID.Value

16. Right-click the table handle to the left of the row and select **Table Footer** to remove the footer row from the table.
17. Select the detail row and in the **Properties window**, set the **BorderStyle** property to **Solid**.

 **Note:** In case you are setting a recursive hierarchy on a Fixed Page Layout, set the **DataSetName** property in the **FixedPage 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. Go to [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Page Reports\CPL\Layouts\Reports\Reels.

 **Note:** In Windows Vista, the path is [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Page Reports\CPL\Layouts\Reports\Reels.

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 **Using the Viewer** for further information.

## Reports with Parameterized Queries

You can create dynamic queries to change the structure of a query at run time. This advanced walkthrough illustrates how to create a simple dynamic query.

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. By default, in ActiveReports, the Reels.mdb file is located at [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data\Reels.mdb.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at runtime.

## Design Time Layout



## Runtime Layout

Movie	In Stock	Store Price
Aladdin	0	18.95
Beverly Hills Cop II	11	18.95
Daddy Day Care	10	27.95
Despicable Me	14	38.95
Jurassic Park	10	23.95
Land Before Time II	12	28.95
Meet the Fockers	24	14.95
One Flew Over the Cuckoo's Nest	7	18.95

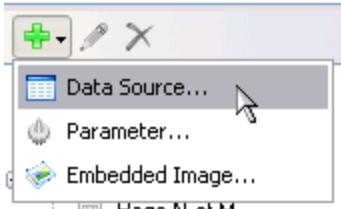
### To add an ActiveReport to the Visual Studio project

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 7 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 **Adding an ActiveReport to a Project** 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

Location

#### Property Value

0in, 0.5in

Size 5.5in, 0.75in

FixedSize (only for FPL 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, 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: 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 CPL 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 **Using the Viewer** for further information.

**Caution:** For an FPL report, you must set the name of the dataset you are using in the report in the **Dataset name** of **FixedPage Dialog** before you preview the report. For example, Products in this walkthrough.

## 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. By default, in ActiveReports, the Reels.mdb file is located at [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data\Reels.mdb.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at runtime.

## Design Time Layout



## Runtime Layout

District Locations		
District Name:	Portland	
State Name:	OR	State:
State #100	W. Linn	OR
State #100	Beaverton	OR
District Name:	Seattle	
State Name:	OR	State:
State #100	Rosemary	WA
District Name:	Vancouver	
State Name:	OR	Province:
State #100	Westminster	BC

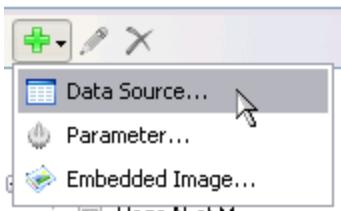
### To add an ActiveReport to the Visual Studio project

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 7 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 [Adding an ActiveReport to a Project](#) 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  
FPL  
reports.

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**

<b>Property Name</b>	<b>Property Value</b>
Value	Store Name
FontWeight	Bold

**TableColumn2**

<b>Property Name</b>	<b>Property Value</b>
Value	City
FontWeight	Bold

**TableColumn3**

<b>Property Name</b>	<b>Property Value</b>
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://ca.maps.yahoo.com/maps_result?csz=" & _City &
"%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 **Using the 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. By default, in ActiveReports Developer, the Reels.mdb file is located in the [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data folder.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at runtime.

## Design Time Layout



## Runtime Layout

Store ID	Net Sales By Store	Net Sales
1001	1	77.385
1002	12	118.02
1003	1	31.99
1004	10	82
1005	21	91.40
1006	11	40
1007	3	36.49
1008	11	37.49
1009	5	26.49

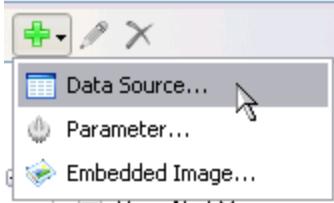
## To add an ActiveReport to the Visual Studio project

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 7 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 **Adding an ActiveReport to a Project** 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. 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 Continuous Page Layout (CPL) 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 **Using the Viewer** for further information.

## Charts in a Page Report

You can create a page report with a chart using the ActiveReports Developer **Chart** control. This walkthrough illustrates how to create a report with a chart.

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 a chart control 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. By default, in ActiveReports Developer, the Reels.mdb file is located in the [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data folder.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at runtime.

## Design Time Layout



## Runtime Layout



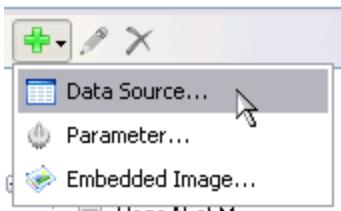
### To add an ActiveReport to the Visual Studio project

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 7 Page Report** and in the Name field, rename the file as **ProfitsByGenre**.
4. Click the **Add** button to open a new fixed page report in the **designer**.

See **Adding an ActiveReport to a Project** 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 **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.



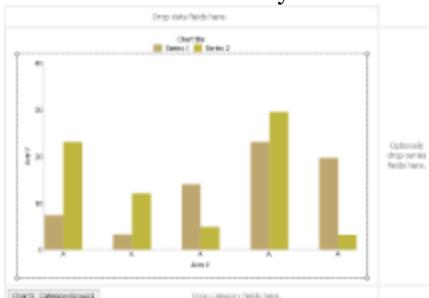
5. On the **Fields** page, add a new field by clicking the Add (+) button above the fields list.
6. Set the **Name** to Month and the **Value** of the new field to `=Fields!SaleDate.Value.Month`  
This parses out the month from the date field.
7. Click **OK** to close the dialog. Your data set and queried fields appear as nodes in the Report Explorer.

#### To add chart control with data and grouping to the report

1. From the toolbox, drag the **Chart** control onto of the report design surface.
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	oin, oin
Size	6.5in, 4.5in

3. Double-click inside the chart 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 Report Explorer, 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 X-Axis label in the chart and at the bottom of the Properties Window, select the **Property dialog** command.
5. 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.

6. Click **OK** to close the dialog.
7. Select the Y-Axis label in the chart and at the bottom of the Properties Window, select the **Property dialog** command.
8. 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.

### **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.

9. 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 **Using the Viewer** for further information.

## Interactive Reports

ActiveReports Developer offers a variety of ways to make reports interactive. Follow these walkthroughs to learn about some of them.

- Bookmarks allow users to follow a link to another part of the report.
- Drilldown Reports allow users to drill down into more detail that is initially hidden.
- Drill-through Reports allow users to drill through into more detail in separate reports.
- Parameters allow users to select which data to use in the report.

## 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. By default, in ActiveReports Developer, the Factbook.rdsx file is located at [User Documents folder]\ComponentOne Samples\ActiveReports Developer \Data\Factbook.rdsx.

 **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 runtime.

## Design Time Layout



## Runtime Layout



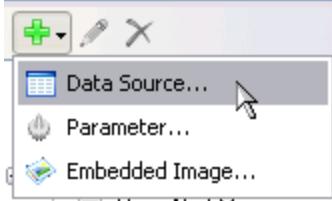
### To add an ActiveReport to a Visual Studio project

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 7 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 [Adding an ActiveReport to a Project](#) 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, which is located in [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data. 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
Name	Database Field	@name
Currency	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 oin, oin and **Size** property to 6.5in, 3in.
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 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
Currency	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

2000                          Size: 1in, 0.25in  
                                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.

<b>Property Name</b>	<b>Property Value</b>
----------------------	-----------------------

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**

<b>Property Name</b>	<b>Property Value</b>
Location	0.125in, 0.5in
Size	0.75in, 0.25in
FontWeight	Bold
Value	Currency:

**TextBox2**

<b>Property Name</b>	<b>Property Value</b>
Location	3.375in, 0.875in
Size	0.5in, 0.25in
TextAlign	Right
Value	2004:

**TextBox3**

<b>Property Name</b>	<b>Property Value</b>
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 **Name** 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 **Name** 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 (**Name**) created in the procedure above.
5. Click **OK** to close the dialog.

## To view the report

Open the report in the Viewer. See **Using the Viewer** for further information.

## 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. By default, in ActiveReports Developer, the Reels.mdb file is located in the [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data folder.

When you complete this walkthrough you get a layout that looks similar to the following at runtime.

## Design Time Layout



## Runtime Layout

Customer	Quantity	Price	Total
Mr. & Mrs. Smith	1	\$21.49	\$21.49
The Pioneers	1	\$13.45	\$13.45
Kidney Beefs	1	\$11.45	\$11.45
			\$54.34
Symbol			\$44.40
Sweeney			\$100.00
Tellaria			\$71.64
Mitchell			\$35.00
			\$1,561.02

## 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 **Using the 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. By default, in ActiveReports Developer, the Reels.mdb file is located in the [User Documents folder]\ComponentOne Samples\ActiveReports Developer \Data folder.

When you complete this walkthrough you get a layout that looks similar to the following at runtime.

## Runtime Layout (main report)



## Runtime Layout (detail report)

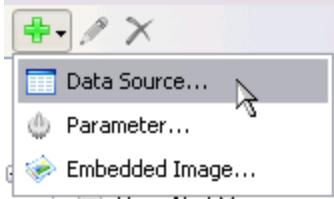


### To create the main report

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 7 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 7 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	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

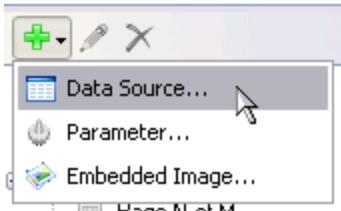
Property Name	Property Value
BorderStyle	Solid
Font	Normal, Arial, 10pt, Bold
TextAlign	Center

### 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 7 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 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 FPL report, set the Dataset name in the FixedPage dialog > General tab to **MovieInfo**. For more information, see **FixedPage Dialog**.

**Caution:** In an FPL report, you may get an error if the Dataset name for the FixedPage is not be specified explicitly.

### 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 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	oin, oin
Name	MovieList
Size	6.5in, iin
FixedSize (only for FPL reports)	6.5in, iin

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.

- 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. On the design surface, select the cell containing the **Title** field inside the table data region and at the bottom of the Properties Window, click the Property dialog command.
2. In the **Textbox** dialog that appears, go to the **Navigation** page.
3. Under **Action**, select **Jump to report** and set the report name MovieDetails.rdlx.
4. 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.

5. Set the **Value** to `=Fields!MovieID.Value`.
6. Click **OK** to close the dialog.

## To view the report

Open the report in the Viewer. See **Using the Viewer** for further information.

## Parameterized Reports

You can create a parameterized report with ActiveReports Developer and provide an "All" choice for users who want to see all of the data as well as the ability to select multiple values for those who want to see data for several items.

This walkthrough illustrates how to create a report with multivalue parameters and an option to select all of the data.

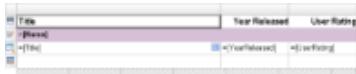
The walkthrough is split up into the following activities:

- 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 **Movie** table from the Reels database. By default, in ActiveReports Developer, the Reels.mdb file is located in the [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data folder.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at runtime.

## Design Time Layout



## Runtime Layout



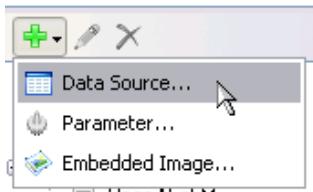
## To add an ActiveReport to the Visual Studio project

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 7 Page Report** and in the Name field, rename the file as **MoviesByProducer**.
4. Click the **Add** button to open a new fixed page report in the **designer**.

See **Adding an ActiveReport to a Project** 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 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 **Producers**. 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 -1 AS ProductionID, "(All)" AS Name
FROM Producers
UNION
SELECT ProductionID, Name
FROM Producers;
```

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. 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: ProductionID
- DataType: Integer
- Text for prompting users for a value: Select a production company.
- Select the check box next to Multivalue to allow users to select more than one production company from the list.

In the **Available Values** tab select From query:

- DataSet: Producers
- Value: ProductionID
- Label: Name

**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. 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 **Movies**. 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 **Param1**.

4. Under **Value** enter `=Parameters!ProductionID.Value`
5. On the **Parameters** page under **Parameter Name** enter **Param2**.
6. Under **Value** enter `=Parameters!ProductionID.Value`
7. On the **Query** page of this dialog, in the **Query** field enter the following SQL query.

### SQL Query

```
SELECT Movie.Title, Movie.YearReleased, Movie.UserRating, Producers.Name
FROM Producers INNER JOIN (Movie INNER JOIN MovieProducers ON Movie.MovieID =
MovieProducers.MovieID) ON Producers.ProductionID = MovieProducers.ProductionID
WHERE (MovieProducers.ProductionID IN (?)) OR (-1 IN (?))
ORDER BY MovieProducers.ProductionID, Movie.YearReleased;
```

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	oin, iin
DataSetName	Movies
FixedSize (only for FPL reports)	6.5in, 7.5in

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 icon to the left of the detail row and select Insert Group.
4. In the **Table - Groups dialog** that appears, under **Expression** select `=Fields!Name.Value` to group all details from each producer.
5. Change the Name to Producer.



**Note:** You cannot change the name of a table group until after you have set the expression.

6. On the **Layout** tab of the Table - Groups dialog, select the check box next to **Repeat group header** to ensure that the header is printed at the top of each page.
7. Clear the check box next to **Include group footer** as we will not be using it for this report.
8. Click **OK** to close the dialog.
9. In the **Report Explorer**, from the **Movies** dataset drag the following fields into the detail row of the table and set their properties as in the following table.

Field	Column	Width
Title	TableColumn1	3.9in
YearReleased	TableColumn2	1.3in
UserRating	TableColumn3	1.3in

10. 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 text to Bold and change the **FontSize** to 11pt.
11. From the **Report Explorer**, the **Movies** dataset drag the **Name** field into the first column of the group header row of the table and set the following properties.

Property Name	Property Value
FontWeight	Bold
BackgroundColor	Thistle



**Tip:** Use the Shift key to select all three text boxes in the group header row, right-click and select Merge Cells to prevent long production company names from wrapping.

### To view the report

- Click the preview tab to view the report at design time.
- OR
- Open the report in the Viewer. See **Using the Viewer** for further information.

## Custom Web Exporting in a Page Report

ActiveReports Developer provides components that allow you to export your reports into several popular formats like PDF, HTML, Excel, Image and Word.

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 and Image Export objects and export a report
- Running the project

### To add an ActiveReport to the Visual Studio project

1. Create a new Visual Studio Web Application project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 7 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.v7  
GrapeCity.ActiveReports.Export.Html.v7  
GrapeCity.ActiveReports.Export.Excel.v7  
GrapeCity.ActiveReports.Export.Word.v7  
GrapeCity.ActiveReports.Export.Image.v7

See **Adding an ActiveReport to a Project** 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.

```
Dim _reportDef As New GrapeCity.ActiveReports.PageReport(New
System.IO.FileInfo(Server.MapPath("") + "\CustomWebExporting.rdlx"))
Dim _reportRuntime As New GrapeCity.ActiveReports.Document.PageDocument(_reportDef)

'Set the rendering extension and render the report.
Dim _renderingExtension As New
GrapeCity.ActiveReports.Export.Pdf.Page.PdfRenderingExtension()
Dim _provider As New GrapeCity.ActiveReports.Rendering.IO.MemoryStreamProvider()
_reportRuntime.Render(_renderingExtension, _provider)

Response.ContentType = "application/pdf"
Response.AddHeader("content-disposition", "inline;filename=MyExport.pdf")
Dim ms As New System.IO.MemoryStream()
CType(_provider.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.

```
GrapeCity.ActiveReports.PageReport _reportDef = new
GrapeCity.ActiveReports.PageReport(new System.IO.FileInfo(Server.MapPath("") +
"\\"CustomWebExporting.rdlx"));
GrapeCity.ActiveReports.Document.PageDocument _reportRuntime = new
GrapeCity.ActiveReports.Document.PageDocument(_reportDef);

// Set the rendering extension and render the report.
GrapeCity.ActiveReports.Export.Pdf.PdfRenderingExtension _renderingExtension = new
GrapeCity.ActiveReports.Export.Pdf.Page.PdfRenderingExtension();
GrapeCity.ActiveReports.Rendering.IO.MemoryStreamProvider _provider = new
GrapeCity.ActiveReports.Rendering.IO.MemoryStreamProvider();
_reportRuntime.Render(_renderingExtension, _provider);

Response.ContentType = "application/pdf";
Response.AddHeader("content-disposition", "inline;filename=MyExport.pdf");
System.IO.MemoryStream ms = new System.IO.MemoryStream();
_provider.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 _reportRuntime.Render(_renderingExtension, _provider) in the
code above with the following
```

```
Dim s As New GrapeCity.ActiveReports.Export.Pdf.Page.Settings()
s.PrintOnOpen = True
_reportRuntime.Render(_renderingExtension, _provider,s)
```

### C# code. Paste INSIDE the Page Load event.

```
// Replace the line _reportRuntime.Render(_renderingExtension, _provider); in the
code above with the following
```

```
GrapeCity.ActiveReports.Export.Pdf.Page.Settings s = new
GrapeCity.ActiveReports.Export.Pdf.Page.Settings();
s.PrintOnOpen = true;
_reportRuntime.Render(_renderingExtension, _provider,s);
```

## 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.

```
Dim _reportDef As New GrapeCity.ActiveReports.PageReport(New
System.IO.FileInfo(Server.MapPath("") + "\\CustomWebExporting.rdlx"))
Dim _reportRuntime As New GrapeCity.ActiveReports.Document.PageDocument(_reportDef)

' Set the rendering extension and render the report.
Dim _renderingExtension As New
GrapeCity.ActiveReports.Export.Html.Page.HtmlRenderingExtension()
Dim _provider As New GrapeCity.ActiveReports.Rendering.IO.MemoryStreamProvider()
Dim s As New GrapeCity.ActiveReports.Export.Html.Page.Settings()
s.Mode = GrapeCity.ActiveReports.Export.Html.Page.RenderMode.Galley
s.MhtOutput = True

_reportRuntime.Render(_renderingExtension, _provider, s)

Response.ContentType = "message/rfc822"
Response.AddHeader("content-disposition", "inline;filename=MyExport.mht")
Dim ms As New System.IO.MemoryStream()
CType(_provider.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.

```
GrapeCity.ActiveReports.PageReport _reportDef = new
GrapeCity.ActiveReports.PageReport(new System.IO.FileInfo(Server.MapPath("") +
"\\CustomWebExporting.rdlx"));
GrapeCity.ActiveReports.Document.PageDocument _reportRuntime = new
GrapeCity.ActiveReports.Document.PageDocument(_reportDef);

// Set the rendering extension and render the report.
GrapeCity.ActiveReports.Export.Html.Page.HtmlRenderingExtension _renderingExtension =
new GrapeCity.ActiveReports.Export.Html.Page.HtmlRenderingExtension();
GrapeCity.ActiveReports.Rendering.IO.MemoryStreamProvider _provider = new
GrapeCity.ActiveReports.Rendering.IO.MemoryStreamProvider();
GrapeCity.ActiveReports.Export.Html.Page.Settings s = new
GrapeCity.ActiveReports.Export.Html.Page.Settings();
s.Mode = GrapeCity.ActiveReports.Export.Html.Page.RenderMode.Galley;

_reportRuntime.Render(_renderingExtension, _provider, s);

Response.ContentType = "message/rfc822";
Response.AddHeader("content-disposition", "inline;filename=MyExport.html");
System.IO.MemoryStream ms = new System.IO.MemoryStream();
_provider.GetPrimaryStream().OpenStream().CopyTo(ms);
Response.BinaryWrite(ms.ToArray());
Response.End();
```

---

## To add code to the Web Form to create the Excel Export object and export a report

 **Note:** Convert your report layout to CPL in order to render it to an Excel format.

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 _reportDef As New GrapeCity.ActiveReports.PageReport(New
System.IO.FileInfo(Server.MapPath("") + "\\CustomWebExporting.rdlx"))
Dim _reportRuntime As New GrapeCity.ActiveReports.Document.PageDocument(_reportDef)

' Set the rendering extension and render the report.
Dim _renderingExtension As New
GrapeCity.ActiveReports.Export.Excel.Page.ExcelTransformationDevice()
Dim _provider As New GrapeCity.ActiveReports.Rendering.IO.MemoryStreamProvider()
Dim s As New GrapeCity.ActiveReports.Export.Excel.Page.Settings()
s.ProtectWorkbookPassword = 123

_reportRuntime.Render(_renderingExtension, _provider, s)

Response.ContentType = "application/vnd.ms-excel"
Response.AddHeader("content-disposition", "inline;filename=MyExport.xls")
Dim ms As New System.IO.MemoryStream()
CType(_provider.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.

```
GrapeCity.ActiveReports.PageReport _reportDef = new
GrapeCity.ActiveReports.PageReport(new System.IO.FileInfo(Server.MapPath("") +
"\\"CustomWebExporting.rdlx"));
GrapeCity.ActiveReports.Document.PageDocument _reportRuntime = new
GrapeCity.ActiveReports.Document.PageDocument(_reportDef);

// Set the rendering extension and render the report.
GrapeCity.ActiveReports.Export.Excel.Page.ExcelTransformationDevice _renderingExtension
= new GrapeCity.ActiveReports.Export.Excel.Page.ExcelTransformationDevice();
GrapeCity.ActiveReports.Rendering.IO.MemoryStreamProvider _provider = new
GrapeCity.ActiveReports.Rendering.IO.MemoryStreamProvider();
GrapeCity.ActiveReports.Export.Excel.Page.Settings s = new
GrapeCity.ActiveReports.Export.Excel.Page.Settings();
s.ProtectWorkbookPassword = "123";

_reportRuntime.Render(_renderingExtension, _provider, s);

Response.ContentType = "application/vnd.ms-excel";
Response.AddHeader("content-disposition", "inline;filename=MyExport.xls");
System.IO.MemoryStream ms = new System.IO.MemoryStream();
_provider.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.

## To write the code in Visual Basic.NET

### Visual Basic.NET code. Paste INSIDE the Page Load event.

```
Dim _reportDef As New GrapeCity.ActiveReports.PageReport(New
System.IO.FileInfo(Server.MapPath("") + "\\CustomWebExporting.rdlx"))
Dim _reportRuntime As New GrapeCity.ActiveReports.Document.PageDocument(_reportDef)

' Set the rendering extension and render the report.
Dim _renderingExtension As New
GrapeCity.ActiveReports.Export.Word.Page.WordRenderingExtension()
Dim _provider As New GrapeCity.ActiveReports.Rendering.IO.MemoryStreamProvider()
Dim s As New GrapeCity.ActiveReports.Export.Word.Page.Settings()
s.UseMhtOutput = True

_reportRuntime.Render(_renderingExtension, _provider, s)

Response.ContentType = "application/msword"
Response.AddHeader("content-disposition", "inline;filename=MyExport.doc")
Dim ms As New System.IO.MemoryStream()
CType(_provider.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.

```
GrapeCity.ActiveReports.PageReport _reportDef = new
GrapeCity.ActiveReports.PageReport(new System.IO.FileInfo(Server.MapPath("") +
"\\CustomWebExporting.rdlx"));
GrapeCity.ActiveReports.Document.PageDocument _reportRuntime = new
GrapeCity.ActiveReports.Document.PageDocument(_reportDef);

// Set the rendering extension and render the report.
GrapeCity.ActiveReports.Export.Word.Page.WordRenderingExtension _renderingExtension =
new GrapeCity.ActiveReports.Export.Word.Page.WordRenderingExtension();
GrapeCity.ActiveReports.Rendering.IO.MemoryStreamProvider _provider = new
GrapeCity.ActiveReports.Rendering.IO.MemoryStreamProvider();
GrapeCity.ActiveReports.Export.Word.Page.Settings s = new
GrapeCity.ActiveReports.Export.Word.Page.Settings();
s.UseMhtOutput = true;

_reportRuntime.Render(_renderingExtension, _provider, s);

Response.ContentType = "application/msword";
Response.AddHeader("content-disposition", "inline;filename=MyExport.doc");
System.IO.MemoryStream ms = new System.IO.MemoryStream();
_provider.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.

```
Dim _reportDef As New GrapeCity.ActiveReports.PageReport(New
```

```
System.IO.FileInfo(Server.MapPath("") + "\\CustomWebExporting.rdlx"))
Dim _reportRuntime As New GrapeCity.ActiveReports.Document.PageDocument(_reportDef)

' Set the rendering extension and render the report.
Dim _renderingExtension As New
GrapeCity.ActiveReports.Export.Image.Page.ImageRenderingExtension()
Dim _provider As New GrapeCity.ActiveReports.Rendering.IO.MemoryStreamProvider()
Dim s As New GrapeCity.ActiveReports.Export.Image.Page.Settings()
s.ImageType = GrapeCity.ActiveReports.Export.Image.Page.Renderers.ImageType.JPEG

_reportRuntime.Render(_renderingExtension, _provider, s)

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(_provider.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.

```
GrapeCity.ActiveReports.PageReport _reportDef = new
GrapeCity.ActiveReports.PageReport(new System.IO.FileInfo(Server.MapPath("") +
"\\CustomWebExporting.rdlx"));
GrapeCity.ActiveReports.Document.PageDocument _reportRuntime = new
GrapeCity.ActiveReports.Document.PageDocument(_reportDef);

// Set the rendering extension and render the report.
GrapeCity.ActiveReports.Export.Image.Page.ImageRenderingExtension _renderingExtension =
new GrapeCity.ActiveReports.Export.Image.Page.ImageRenderingExtension();
GrapeCity.ActiveReports.Rendering.IO.MemoryStreamProvider _provider = new
GrapeCity.ActiveReports.Rendering.IO.MemoryStreamProvider();
GrapeCity.ActiveReports.Export.Image.Page.Settings s = new
GrapeCity.ActiveReports.Export.Image.Page.Settings();
s.ImageType = GrapeCity.ActiveReports.Export.Image.Renderers.ImageType.JPEG;

_reportRuntime.Render(_renderingExtension, _provider, s);

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
_provider.GetSecondaryStreams()[0].OpenStream().CopyTo(ms);
Response.BinaryWrite(ms.ToArray());
Response.End();
```

---

## To run the project

Press **F5** to run the project.

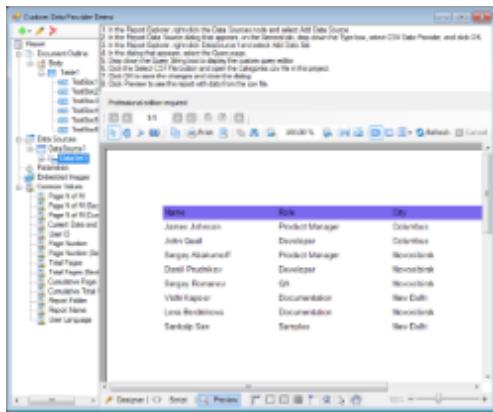
## Custom Data Provider

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 7 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.
9. From the toolbox ActiveReports 7 tab, drag a Designer control and drop it on the empty part of the form.
10. Set the **Dock** property to **Fill**, then right-click the Designer control on the form and select **Bring to front**.
11. Select the ReportExplorer control and in the Properties window, drop down the **ReportDesigner** property and select **Designer1**.
12. 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 7 Page 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. Click on the report design surface to show the **Report** menu, drop down that menu, and select **Convert to CPL report**. (This way all of our data shows on one page.)
5. From the ActiveReports 7 Page Report Toolbox, drag a Table report control onto the report.

 **Note:** In case you are still working on FPL layout, set the **FixedSize** property of the Table control to display all data on one page.

6. Click inside the table to reveal the table adorners, then right-click the table adorer to the left of the footer row and select **Delete Rows**. The footer row is removed from the table.
7. 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 7.)

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	

8. In the Report Explorer, select the Table1 node and in the Properties window, set the **Location** property to 1in, 1in and the **Size** property to **6in, 0.5in** to make the table wide enough to see all of the data.
9. 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. Right-click the CustomDataProvider project and select **Add Reference**, and in the Add Reference dialog that appears, on the .NET tab, select the following references to add, holding down the **Ctrl** button to select multiple references.
  - GrapeCity ActiveReports Developer 7 (GrapeCity.ActiveReports.v7)
  - GrapeCity ActiveReports Extensibility Library (GrapeCity.ActiveReports.Extensibility.v7)

- System.Drawing
  - System.Windows.Forms
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
 End Structure
End Namespace
```

```

 ' 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

' Converts 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 or
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
 End Try
End Function
```

```
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
 IDataRecord.GetFieldType
 If i > _columns.Length - 1 Then
 Return Nothing
 End If
 End Function

#End Region
```

```
 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 (""),
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;
 }

 // Converts 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 or 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. (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.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 StreamReader(_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 StreamReader(_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.

**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(localizedName As String)
 _localizedName = localizedName
 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.
 End Class

```

```
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
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 CSVDataProvider
 Public NotInheritable Class QueryEditor
 Inherits UITypeEditor
 Dim path = ""
 Friend WithEvents btn As New Button()
 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)

 btn.Text = "Select CSV File..."
 Dim pdg = btn.Padding
 pdg.Bottom += 2
 btn.Padding = pdg

 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 path
 End Function

 Private Sub btn_Click(sender As System.Object, e As System.EventArgs)
Handles btn.Click
 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
 End Sub
 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 =
openDlg.FileName;
 }
 };
 edSvc.DropDownControl(btn);

 if (string.IsNullOrEmpty(path)) return string.Empty;
 if (!File.Exists(path)) return string.Empty;

 return path;
 }
 }
}
```

```
 }
 }
}
```

- 
2. In the Solution Explorer, right-click the CustomDataProviderDemo project and select **Add Reference**. In the Add Reference 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 comprise of scenarios to introduce the key features of code-based and XML-based section reports.

### **Basic Data Bound Reports**

### **Basic XML-Based Reports (RPX)**

### **Address Labels**

### **Columnar Reports**

### **Overlaying Reports (Letterhead)**

### **Chart Walkthroughs**

### **Custom Web Exporting (Std Edition)**

### **Custom HTML Outputer**

### **Group On Unbound Fields**

### **Subreport Walkthroughs**

### **Mail Merge with RichText**

### **Run Time or Ad Hoc Reporting**

### **Layout Files with Embedded Script**

### **Custom Web Exporting**

### **Custom HTML Outputer**

## Basic Data Bound Reports

In ActiveReports Developer, 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.

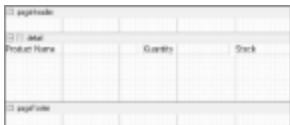
The walkthrough is split up into the following activities:

- 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. By default, in ActiveReports, the Northwind.mdb file is located at [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data\Nwind.mdb.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at runtime.

## Design Time Layout



## Runtime Layout

Chai	'10 boxes x 20 bags	39
Chang	24 - 12 oz bottles	17
Aristed Syrup	12 - 600 ml bottles	13
Chat Avenue Chai Spice	40 - 8 oz jars	53
Chat Avenue Green Tea	36 boxes	9
Chengman Honey	12 - 1 lb jars	120
Florida Orange Sweet Spread	12 - 1 lb jars	15
Northwoods Cranberry Sauce	12 - 12 oz jars	6
Maria Rose Toffee	18 - 900 g jars	29
Bitter	12 - 200 ml bottles	31
Caramel Cakes	1 kg bag	22
Giant Marathwa La Patisse	12 - 500 g bags	96
Ketchup	2 kg bags	24
Total	40 - 1000 g pieces	36

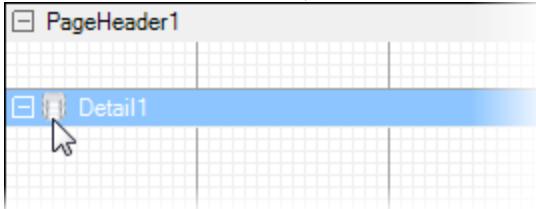
### To add an ActiveReport to the Visual Studio project

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 7 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 [Adding an ActiveReport to a Project](#) 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 7 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
TextField	ProductName
Text	Product Name
Location	oin, oin
Size	2.3in, 0.2in

#### TextBox2

Property Name	Property Value
TextField	QuantityPerUnit
Text	Quantity
Location	2.4in, oin
Size	1.5in, 0.2in

#### TextBox3

Property Name	Property Value
TextField	UnitsInStock
Text	Stock
Location	4in, oin
Size	1in, 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 **Using the Viewer** for further information.

## Subreport Walkthroughs

Follow step-by-step tutorials as you create Visual Studio projects using the ActiveReports Developer **Subreport** control. You can use the subreport control to embed a report within another report. Subreports are executed each time the parent section (i.e. the section in which the Subreport control is placed) is printed. Some ways to use subreports include:

- Repeating a group of relational data (for example, a list of orders in the main report, with ordered items in the subreport)
- Using multiple data sources within a report
- Creating multiple detail sections within a report

#### This section contains information about how to:

##### **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.

##### **Subreports with XML Data**

Learn how to use XML data with subreports.

## Subreports with Run-Time Data Sources

ActiveReports Developer 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.

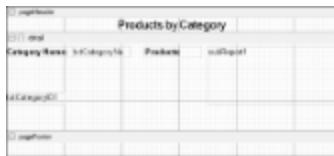
This walkthrough is split up into the following activities:

- 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. By default, in ActiveReports Developer, the NWind.mdb file is located in the [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Samples\Data folder.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at runtime.

## Design Time Layout



## Runtime Layout



### To add an ActiveReport to the Visual Studio project

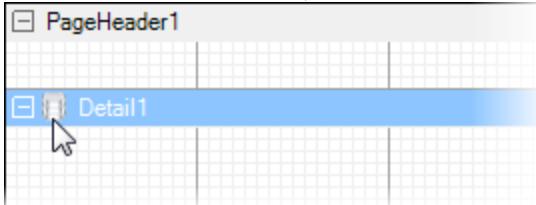
1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 7 Section Report (code-based)** and in the Name field, rename the file as **rptMain**.
4. Click the **Add** button to open a new section report in the **designer**.
5. From the **Project** menu, select **Add New Item**.
6. In the Add New Item dialog that appears, select **ActiveReports 7 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 **Adding an ActiveReport to a Project** 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 **Using the 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.

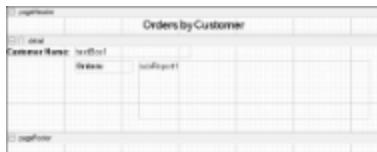
This walkthrough is split up into the following activities:

- 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 tables from the NWind database. By default, in ActiveReports Developer, the NWind.mdb file is located in the [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Samples\Data folder.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at runtime.

## Design Time Layout



## Runtime Layout



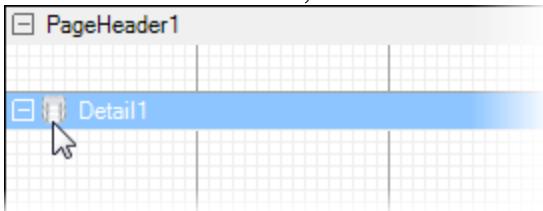
## To add an ActiveReport to the Visual Studio project

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 7 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 7 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 **Adding an ActiveReport to a Project** 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. (The default installation path is C:\Users\YourUserName\Documents\ComponentOne Samples\ActiveReports Developer 7\Data\customer.xml).
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 detail section and in the Properties window, set the **CanShrink** property to **True** to eliminate white space.
3. 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

4. From the toolbox, drag the controls onto the detail section and in the Properties window, set the properties of each control as follows.

### TextBox1

<b>Property Name</b>	<b>Property Value</b>
TextField	NAME
Location	1.2, 0 in
Size	2, 0.2 in

**Label1**

<b>Property Name</b>	<b>Property Value</b>
Text	Customer Name:
Location	0, 0 in
Size	1.2, 0.2 in
Font Bold	True

**Label2**

<b>Property Name</b>	<b>Property Value</b>
Text	Orders:
Location	1.2, 0.25 in
Size	1, 0.2 in
Font Bold	True

**Subreport**

<b>Property Name</b>	<b>Property Value</b>
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.

<b>Property Name</b>	<b>Property Value</b>
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**

<b>Property Name</b>	<b>Property Value</b>
TextField	TITLE
Name	txtTitle
Location	0, 0 in

Size 2.9, 0.2 in

#### TextBox2

Property	Property Value
Name	
TextField	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 **Using the Viewer** for further information.

## Chart Walkthroughs

Charts add quick visual impact to your reports, and allow data to be readily grasped even by casual readers. With a built-in chart control, ActiveReports Developer makes it easy to provide premium reporting without the need to purchase extra tools.

### **Bar Chart**

Describes how to create a bar chart which compares items across categories.

### **3D Pie Chart**

Describes how to create a three dimensional pie chart which shows how the percentage of each data item contributes to a total percentage.

### **Financial Chart**

Describes how to create a financial chart which lets you plot high, low, opening, and closing prices.

### **Simple Unbound Chart**

Describes how to create a simple unbound chart.

## Bar Chart

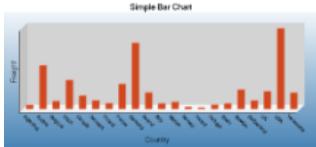
Bar charts are useful in comparing items across categories. This walkthrough illustrates how to create a simple bar chart using the ActiveReports Developer chart control.

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. By default, in ActiveReports, the Northwind.mdb file is located at [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data\NWIND.mdb.

When you complete this walkthrough you get a layout that looks similar to the following at runtime.



### 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 7 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 **Adding an ActiveReport to a Project** for information on adding different report layouts.

### To add the Chart control to the report

1. From the toolbox, drag the **ChartControl** 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, 0 in
Size	6.5, 3.5 in

4. 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

- a. Click the **Axes** bar on the left to expand it.
- b. 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**.
- c. On the **Labels** tab, select the **Staggered Labels** checkbox to avoid overlapping labels and set the **Text angle** property to **45**.



- d. 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

- a. Click the **Titles** bar on the left to expand it. In the list of titles, the **header** is selected by default.
- b. In the **Caption** textbox, type **Simple Bar Chart** and increase the **Font size** to **14**.
- c. 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

- a. Click the **Series** bar on the left. The **Series1** is selected by default.
- b. In the **Data Binding** box, set **X (Name)** to **ShipCountry**, and set **Y** to **FreightSum**.
- c. 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

- a. Click the **Legend** bar on the left to expand it. The **defaultLegend** is selected by default.
- b. 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 [Using the 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

**Note:** This walkthrough uses the Northwind database. By default, in ActiveReports, the Northwind.mdb file is located at [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data\NWind.mdb.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at runtime.



#### 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 7 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 [Adding an ActiveReport to a Project](#) for information on adding different report layouts.

## To add the Chart control to the report

1. From the toolbox, drag the **ChartControl** 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	oin, oin
Size	6.5in, 3.5in

4. In the **Report Explorer**, select **Detail1** and go to the properties window to set the **Height** property to **3.5**.

## 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 **Using the Viewer** for further information.

## Financial Chart

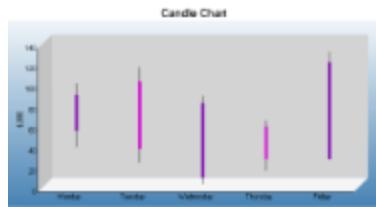
Financial charts are useful for displaying stock information using High, Low, Open and Close values. This walkthrough illustrates how to create a Candle chart.

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

 **Note:** This walkthrough uses the Northwind database. By default, in ActiveReports, the Northwind.mdb file is located at [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data\NWIND.mdb.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at runtime.



### 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 7 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 **Adding an ActiveReport to a Project** for information on adding different report layouts.

### To add the Chart control to the report

1. From the toolbox, drag the **ChartControl** 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	0in, 0in
Size	6.5in, 3.5in

4. In the **Report Explorer**, select **Detail1** and go to the properties window to set the **Height** property to **3.5**.

### 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**
  - a. Under **AxisX** properties, in the **Title** property delete the default text.
  - b. 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.
  - c. In the **Array Data Editor** that appears, enter the following into the editor, each item on a separate line:
    - Monday
    - Tuesday
    - Wednesday
    - Thursday
    - Friday
  - d. Click the **OK** button to return to the **AxisBase Collection Editor**.
  - e. 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

- f. 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**

- a. Under **header properties**, set the **Text** property to **Candle Chart**.
  - b. Expand the **Font** property and set **Font Size** to **14**.
  - c. 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**
  - a. In the Properties window, set the **Visible** property to **False**.
  - b. 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 **Using the Viewer** for further information.

## Unbound Chart

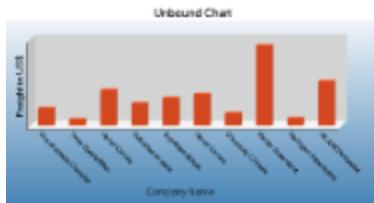
The Chart control allows you to bind charts to any type of data source, including arrays. You can create a chart without setting its data source and load the data into the control at run time. This walkthrough illustrates how to create a simple unbound chart.

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. By default, in ActiveReports, the Northwind.mdb file is located at [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data\NWIND.mdb.

When you complete this walkthrough you get a layout that looks similar to the following at runtime.



## 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 7 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 **Adding an ActiveReport to a Project** for information on adding different report layouts.

## To add the Chart control to the report

1. From the toolbox, drag the **ChartControl** 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	oin, oin
Size	6.5in, 3.5in

4. In the **Report Explorer**, select **Detail** and go to the properties window to set the **Height** property to **3.5**.

#### 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 = "C:\Documents and Settings\YourUserName\My
Documents\GrapeCity\ActiveReports Developer 7\Samples\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 = "C:\\Documents and Settings\\YourUserName\\My
Documents\\GrapeCity\\ActiveReports Developer 7\\Samples\\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 **Using the Viewer** for further information.

## Custom Web Exporting (Std Edition)

ActiveReports Developer 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 Reference dialog that appears, select the following references and click **OK** to add them to your

project.

```
GrapeCity.ActiveReports.Export.Pdf.v7
GrapeCity.ActiveReports.Export.Html.v7
GrapeCity.ActiveReports.Export.Excel.v7
GrapeCity.ActiveReports.Export.Word.v7
GrapeCity.ActiveReports.Export.Image.v7
```

4. In the Add New Item window that appears, select the **ActiveReports 7 Section Report (code-based)** template, set the report's name to **SectionReport1**, and click the **Add** button.
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.Xls.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();
NewActiveReport1 rpt = 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.Xls.Section.XlsExport1 XlsExport1 = new
GrapeCity.ActiveReports.Export.Xls.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.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();
NewActiveReport1 rpt = 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.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.Rtf.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();
NewActiveReport1 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.Rtf.Section.RtfExport rtfExport1 = new
GrapeCity.ActiveReports.Export.Rtf.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.Text.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();
NewActiveReport1 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.Text.Section.TextExport textExport1 = new
GrapeCity.ActiveReports.Export.Text.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 Outputer

You can create a custom HTML outputer for your ActiveReports Developer ASP.NET Web Application.

 **Note:** You cannot create a custom HTML outputer 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 outputer
- 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 outputer

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 **MyCustomHtmlOutputer** 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 **MyCustomHtmlOutputer** class.

#### C# code.

```
public class MyCustomHtmlOutputer: 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();
 //Build 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 ActiveReport to the project, and name it **rptCustHTML**.

2. 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.HtmlExport1 = new
GrapeCity.ActiveReports.Export.Html.Section.HtmlExport();

//Export the report to HTML in this session's webcache
MyCustomHtmlOutputter outputter = new MyCustomHtmlOutputter(this.Context);
this.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.

## Basic XML-Based Reports (RPX)

ActiveReports Developer 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. By default, in ActiveReports, the NWind.mdb file is located in the [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data folder.

When you have finished this walkthrough, you get a report that looks similar to the following at design time and at runtime.

## Design Time Layout



## Runtime Layout

Chair	100x100 x 20 height	30
Almond	24 - 100x100 height	45
Almond Styling	12 - 100x100 height	12
Antique White	40 - 100x100 height	40
Chair Alpine Granite	30x100 height	6
Chair Alpine Granite Small	12 - 8x100 height	12
Chair Baltic Granite	12 - 100x100 height	12
Chair Baltic Granite Small	12 - 8x100 height	12
Chair Black Granite	12 - 100x100 height	12
Chair Black Granite Small	12 - 8x100 height	12
Chair Black Nickel	10 - 100x100 height	20
Chair Black Nickel Small	10 - 8x100 height	20
Chair Dark Oak	12 - 100x100 height	12
Chair Dark Oak Small	12 - 8x100 height	12
Chair Mahogany	10 - 100x100 height	20
Chair Mahogany Small	10 - 8x100 height	20
Chair Oak	12 - 100x100 height	12
Chair Oak Small	12 - 8x100 height	12
Chair Teak	12 - 100x100 height	12
Chair Teak Small	12 - 8x100 height	12
Chair Walnut	10 - 100x100 height	20
Chair Walnut Small	10 - 8x100 height	20
Table	2 - 100x100 height	24
Table Small	2 - 8x100 height	24

### 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 7 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 **Adding an ActiveReport to a Project** for information on adding different report layouts.

### To create a layout for the report

1. In the Visual Studio toolbox, expand the **ActiveReports 7 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	oin, oin
Size	2.3in, 0.2in

### TextBox2

Property Name	Property Value
---------------	----------------

TextField	QuantityPerUnit
Text	Quantity
Location	2.4in, 0in
Size	1.5in, 0.2in

**TextBox3**

Property Name	Property Value
TextField	UnitsInStock
Text	Stock
Location	4in, 0in
Size	1in, 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=C:\Users\[User Folder]\Documents\ComponentOne Samples\ActiveReports
 Developer 7\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=C:\Users\[User Folder]\Documents\ComponentOne Samples\ActiveReports Developer 7\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 Detail1_Format
if b then
 Me.Detail1.BackColor = Color.AliceBlue
 b= false
else
 me.Detail1.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 Detail1_Format()
{
 if(color)
 {
 this.Detail1.BackColor = System.Drawing.Color.AliceBlue;
 color = false;
```

```
 }
 else
 {
 this.Detail1.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);
```

---

## Layout Files with Embedded Script

ActiveReports Developer 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 be 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**

Describes how to embed script in a simple stand-alone report.

### **Script for Subreports**

Describes how to embed script to pass a parameter to a subreport.

## Script for Simple Reports

ActiveReports Developer 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:

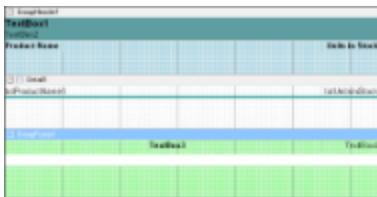
- Temporarily connecting the report to a data source
- Adding controls to a report to display data
- Adding scripting to supply data for the controls
- Saving the report to an RPX file

 **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.

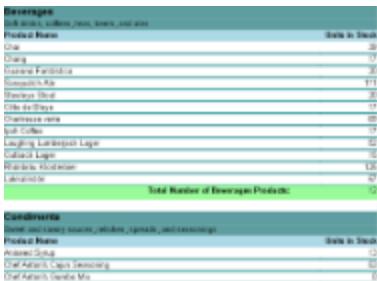
 **Note:** This walkthrough uses the the Northwind database. By default, in ActiveReports, the NWind.mdb file is located in [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data folder.

When you have finished this walkthrough, you will have a report that looks similar to the following at design time and at runtime.

## Design Time Layout



## Runtime 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 7 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 **Adding an ActiveReport to a Project** 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

- 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

- 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.
- 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.
- Increase the group footer section's height so that you have room to work.
- 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
TextField	TotalLabel
Location	2.5, 0 in
Size	3, 0.2 in
Font Bold	True

**TextBox2**

Property Name	Property Value
TextField	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=C:\Users\[User Folder]\Documents\ComponentOne Samples\ActiveReports Developer 7\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]\Documents\ComponentOne Samples\ActiveReports Developer 7\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 **Using the Viewer** for further information.

## Script for Subreports

ActiveReports Developer 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:

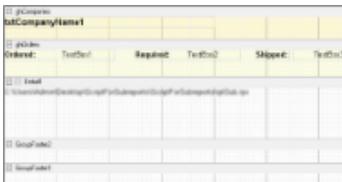
- Temporarily connecting the main report to a data source
- Connecting the subreport to a data source
- Adding controls to each report to display data
- Adding the scripting code for rptMain
- Viewing the report

 **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. By default, in ActiveReports, the Northwind.mdb file is located at [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\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 runtime.

## Design Time Layout (main report)



## Runtime Layout (main report)

Alfredo Futurista		Required	03/20/05	Shipped	03/20/05
Product Name		Quantity	Unit Price	Amount	
Alfredo's Stained	10	\$10.00	\$100.00	0.00%	
Coffee Beans	2	\$10.00	\$20.00	2.00%	
Strawberries	2	\$10.00	\$20.00	2.00%	
<b>Total:</b>				<b>\$140.00</b>	
Product Name	Quantity	Unit Price	Amount		
Alfredo's Special	20	\$20.00	\$400.00	0.00%	
<b>Total:</b>				<b>\$400.00</b>	

## 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 7 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 [Adding an ActiveReport to a Project](#) 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 [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 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 7 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 [Adding an ActiveReport to a Project](#) 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.

- 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
```

- Click **OK** to save the data source and return to the report design surface.

#### To create a layout for the main report

- Right-click the design surface of rptMain and select **Insert** then **Group Header/Footer** to add group header and footer sections to the report.
- 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

- 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

- 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.
- 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

- 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**

TextField	RequiredDate
Location	3.5, 0 in
Size	1, 0.2 in
OutputFormat	MM/dd/yy

**TextBox3****Property Name**      **Property Value**

TextField	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:
FontStyle	Bold

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	C:\full project path\rptSub.rpx
Name	SubReport1
Size	6.5, 1 in
Location	0, 0 in
GroupKeepTogether	All

KeepTogether      True

#### 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**

<b>Property Name</b>	<b>Property Value</b>
X1	4.3
X2	4.3
Y1	0
Y2	0.2

**Line3**

<b>Property Name</b>	<b>Property Value</b>
X1	5.45
X2	5.45
Y1	0
Y2	0.2

**Line4**

<b>Property Name</b>	<b>Property Value</b>
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.

<b>Property Name</b>	<b>Property Value</b>
BackColor	Gainsboro
CanShrink	True

6. From the toolbox, drag four TextBox controls onto onto the Detail section and set the properties as follows.

**TextBox1**

<b>Property Name</b>	<b>Property Value</b>
DataField	ProductName
Location	0, 0 in
Size	3.15, 0.2 in
Alignment	Left

**TextBox2**

<b>Property Name</b>	<b>Property Value</b>
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=C:\Users\[User
Folder]\Documents\ComponentOne Samples\ActiveReports Developer 7\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 Detail1_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 Detail1_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=C:\Users\[User
Folder]\Documents\ComponentOne Samples\ActiveReports Developer 7\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 **Using the Viewer** for further information on how to load the xml-based section report onto the viewer.

## Address Labels

ActiveReports Developer 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. By default, in ActiveReports, the NWind.mdb file is located in the [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data folder.

When you have finished this walkthrough, you get a report that looks similar to the following at design time and at runtime.

## Design Time Layout



## Runtime Layout



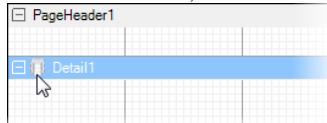
### To add an ActiveReport to the Visual Studio project

- Create a new Visual Studio project.
- From the **Project** menu, select **Add New Item**.
- In the Add New Item dialog that appears, select **ActiveReports 7 Section Report (code-based)** and in the Name field, rename the file as **rptLabels**.
- Click the **Add** button to open a new section report in the **designer**.

See **Adding an ActiveReport to a Project** for information on adding different report layouts.

### To connect the report to a data source

- On the detail section band, click the Data Source Icon.



- In the Report Data Source dialog that appears, on the **OLE DB** tab, next to Connection String, click the Build button.
- 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.
- 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 [Using the Viewer](#) for further information.

## Columnar Reports

ActiveReports Developer 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**.

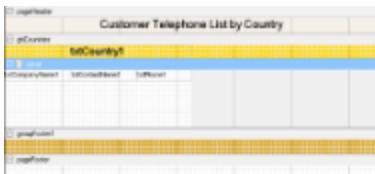
This walkthrough illustrates how to create a simple report using columns, and is split up into the following activities:

- 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. By default, in ActiveReports, the Northwind.mdb file is located at [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data\NWind.mdb.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at runtime.

## Design Time Layout



## Runtime Layout

Customer Telephone List by Country			
	Country	Phone	Address
Promo객인	Argentina	(011) 555-0002	Guadalajara
Marketing	France	(011) 555-0003	Montevideo
Marketing	France	(011) 555-0004	Paris
Marketing	France	(011) 555-0005	Paris
Marketing	France	(011) 555-0006	Paris
Marketing	France	(011) 555-0007	Paris
Marketing	France	(011) 555-0008	Paris
Marketing	France	(011) 555-0009	Paris
Marketing	France	(011) 555-0010	Paris
Marketing	France	(011) 555-0011	Paris
Marketing	France	(011) 555-0012	Paris
Marketing	France	(011) 555-0013	Paris
Marketing	France	(011) 555-0014	Paris
Marketing	France	(011) 555-0015	Paris
Marketing	France	(011) 555-0016	Paris
Marketing	France	(011) 555-0017	Paris
Marketing	France	(011) 555-0018	Paris
Marketing	France	(011) 555-0019	Paris
Marketing	France	(011) 555-0020	Paris
Marketing	France	(011) 555-0021	Paris
Marketing	France	(011) 555-0022	Paris
Marketing	France	(011) 555-0023	Paris
Marketing	France	(011) 555-0024	Paris
Marketing	France	(011) 555-0025	Paris
Marketing	France	(011) 555-0026	Paris
Marketing	France	(011) 555-0027	Paris
Marketing	France	(011) 555-0028	Paris
Marketing	France	(011) 555-0029	Paris
Marketing	France	(011) 555-0030	Paris
Marketing	France	(011) 555-0031	Paris
Marketing	France	(011) 555-0032	Paris
Marketing	France	(011) 555-0033	Paris
Marketing	France	(011) 555-0034	Paris
Marketing	France	(011) 555-0035	Paris
Marketing	France	(011) 555-0036	Paris
Marketing	France	(011) 555-0037	Paris
Marketing	France	(011) 555-0038	Paris
Marketing	France	(011) 555-0039	Paris
Marketing	France	(011) 555-0040	Paris
Marketing	France	(011) 555-0041	Paris
Marketing	France	(011) 555-0042	Paris
Marketing	France	(011) 555-0043	Paris
Marketing	France	(011) 555-0044	Paris
Marketing	France	(011) 555-0045	Paris
Marketing	France	(011) 555-0046	Paris
Marketing	France	(011) 555-0047	Paris
Marketing	France	(011) 555-0048	Paris
Marketing	France	(011) 555-0049	Paris
Marketing	France	(011) 555-0050	Paris
Marketing	France	(011) 555-0051	Paris
Marketing	France	(011) 555-0052	Paris
Marketing	France	(011) 555-0053	Paris
Marketing	France	(011) 555-0054	Paris
Marketing	France	(011) 555-0055	Paris
Marketing	France	(011) 555-0056	Paris
Marketing	France	(011) 555-0057	Paris
Marketing	France	(011) 555-0058	Paris
Marketing	France	(011) 555-0059	Paris
Marketing	France	(011) 555-0060	Paris
Marketing	France	(011) 555-0061	Paris
Marketing	France	(011) 555-0062	Paris
Marketing	France	(011) 555-0063	Paris
Marketing	France	(011) 555-0064	Paris
Marketing	France	(011) 555-0065	Paris
Marketing	France	(011) 555-0066	Paris
Marketing	France	(011) 555-0067	Paris
Marketing	France	(011) 555-0068	Paris
Marketing	France	(011) 555-0069	Paris
Marketing	France	(011) 555-0070	Paris
Marketing	France	(011) 555-0071	Paris
Marketing	France	(011) 555-0072	Paris
Marketing	France	(011) 555-0073	Paris
Marketing	France	(011) 555-0074	Paris
Marketing	France	(011) 555-0075	Paris
Marketing	France	(011) 555-0076	Paris
Marketing	France	(011) 555-0077	Paris
Marketing	France	(011) 555-0078	Paris
Marketing	France	(011) 555-0079	Paris
Marketing	France	(011) 555-0080	Paris
Marketing	France	(011) 555-0081	Paris
Marketing	France	(011) 555-0082	Paris
Marketing	France	(011) 555-0083	Paris
Marketing	France	(011) 555-0084	Paris
Marketing	France	(011) 555-0085	Paris
Marketing	France	(011) 555-0086	Paris
Marketing	France	(011) 555-0087	Paris
Marketing	France	(011) 555-0088	Paris
Marketing	France	(011) 555-0089	Paris
Marketing	France	(011) 555-0090	Paris
Marketing	France	(011) 555-0091	Paris
Marketing	France	(011) 555-0092	Paris
Marketing	France	(011) 555-0093	Paris
Marketing	France	(011) 555-0094	Paris
Marketing	France	(011) 555-0095	Paris
Marketing	France	(011) 555-0096	Paris
Marketing	France	(011) 555-0097	Paris
Marketing	France	(011) 555-0098	Paris
Marketing	France	(011) 555-0099	Paris
Marketing	France	(011) 555-0100	Paris

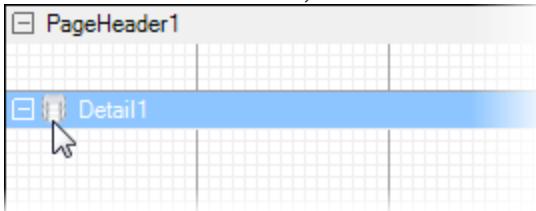
### To add an ActiveReport to the Visual Studio project

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 7 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 [Adding an ActiveReport to a Project](#) 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
FontSize	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
FontSize	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
Field	CompanyName
Location	0, 0 in
Size	1.15, 0.2 in
Font Size	8pt

#### TextBox2

Property Name	Property Value
Field	ContactName
Location	1.15, 0 in
Size	1.15, 0.2 in

Font Size 8pt

### TextBox3

Property Name	Property Value
Field	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 **Using the Viewer** for further information.

## Overlaying Reports (Letterhead)

ActiveReports Developer allows you to overlay static report formats over data reports. This walkthrough illustrates how to overlay an ActiveReport with a static letterhead report.

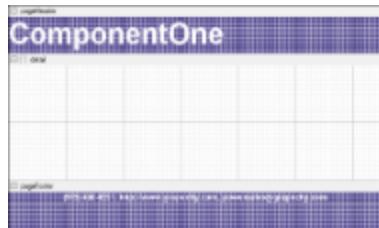
This walkthrough is split up into the following activities:

- 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. By default, in ActiveReports, the Northwind.mdb file is located at [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data\NWIND.mdb.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at runtime.

## Design Time Layout (rptLetterhead)



## Design Time Layout (rptData)



## Runtime Layout



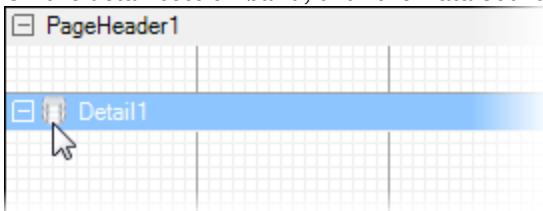
### To add an ActiveReport to the Visual Studio project

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 7 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 7 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 **Adding an ActiveReport to a Project** 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. Right-click the report and select **Insert > GroupHeader/Footer** to add group header and group footer sections.
3. 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

4. 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

- Click the **Detail** section and in the Properties window, set the properties as follows.

Property Name	Property Value
BackColor	LightGray
CanShrink	True

- 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
---------------	----------------

TextField	City
Size	1, 0.2 in
Location	5.5, 0.2 in

7. 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	ComponentOne

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	(919) 460-4551, <a href="http://www.grapecity.com">http://www.grapecity.com</a> , <a href="mailto:powersales@grapecity.com">powersales@grapecity.com</a>

#### 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

- Click the preview tab to view the report at design time.
- OR
- Open the report in the Viewer. See **Using the Viewer** for further information.

## Group On Unbound Fields

ActiveReports Developer 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:

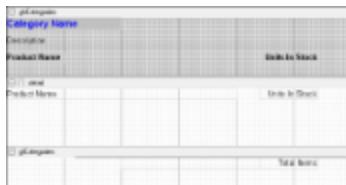
- Adding code to connect the report to a data source
- Adding controls to contain the data
- Using the DataInitialize event to add fields to the report's fields collection
- Using the FetchData event to populate the report fields

- Adding code to close the connection to the data source
- Viewing the report

 **Note:** This walkthrough uses the Northwind database. By default, in ActiveReports, the Northwind.mdb file is located at [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data\NWIND.mdb.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at runtime.

## Design Time Layout



## Runtime 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 7 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 [Adding an ActiveReport to a Project](#) 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=C:\Users\[User
Folder]\Documents\ComponentOne Samples\ActiveReports Developer 7\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=C:\Users\[User
Folder]\Documents\ComponentOne Samples\ActiveReports Developer 7\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
TextField	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
TextField	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

- 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
TextField	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	o
Y2	o

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 **Using the Viewer** for further information.

## Mail Merge with RichText

ActiveReports Developer 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. By default, in ActiveReports, the Northwind.mdb file is located at [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data\NWind.mdb.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at runtime.

## Design Time Layout



## Runtime Layout



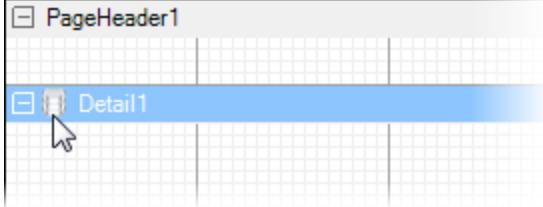
### To add an ActiveReport to the Visual Studio project

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 7 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 **Adding an ActiveReport to a Project** 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. 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

3. 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

4. On the design surface of the report, select the detail section and in the Properties window, set the **CanShrink** property to **True**.
5. 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

6. 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	ComponentOne
Font Size	36
Font Bold	True

7. 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.

- 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

9. 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	MM/dd/yy

**TextBox3 (Subtotal)**

Property Name	Property Value
Location	4.375, 0 in
Size	1, 0.2 in
OutputFormat	Currency
Alignment	Right

10. 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

11. From the toolbox, drag a **Label** control to the pageFooter section and in the Properties window, set the properties as follows.

**Label**

Property Name	Property Value
Location	0, 0 in

Size	6.5, 0.2 in
Text	GrapeCity, 401 Parkplace, Suite 411, Kirkland, WA 98033
Alignment	Center

### 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
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 [Using the Viewer](#) for further information.

## Run Time or Ad Hoc Reporting

ActiveReports Developer allows objects, controls and the data source to be completely accessible at run time. These properties can be modified to provide a dynamic view of your report.

### **Run Time Layouts**

Describes how to create and modify report layouts dynamically.

### **Run Time Data Sources**

Describes how to change the report data source at run time using the ReportStart event.

## Run Time Layouts

ActiveReports Developer 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:

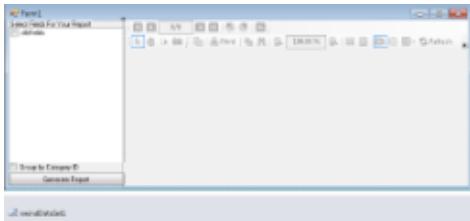
- Adding an ActiveReport to the Visual Studio project

- 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

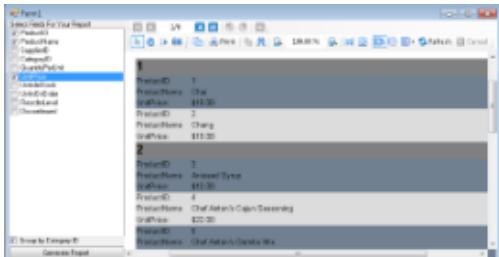
 **Note:** This walkthrough uses the NWind database. By default, in ActiveReports Developer, the Nwind.mdb file is located in the [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data folder.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at runtime.

## Design Time Layout (Windows form)



## Runtime 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 7 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 **Adding an ActiveReport to a Project** 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

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

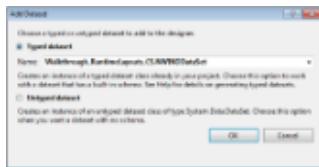
**To generate a dataset for the Form**

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 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 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.btnAddRep.Enabled = False
Else
 Me.btnAddRep.Enabled = True
End If
```

---

## To write the code in C#

### C# code. Paste INSIDE the SelectedIndexChanged event.

```
if(this.clbFields.CheckedItems.Count>0)
{
 this.btnAddRep.Enabled = true;
}
else
{
 this.btnAddRep.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 **Using the Viewer** for further information.

## Run Time Data Sources

ActiveReports allows you to change the data source of a report at run time. This walkthrough illustrates how to find the location of the sample database file on the user's computer and connect the report to it 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 find the database path
- 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. By default, in ActiveReports, the Northwind.mdb file is located at [User Documents folder]\ComponentOne Samples\ActiveReports Developer 7\Data\NWIND.mdb.

When you complete this walkthrough you get a layout that looks similar to the following at design time and at runtime.

## Design Time Layout



## Runtime Layout

1	Ghi	39	E	\$10.00
25	Shelford Blak	22	E	\$10.00
28	Charcoal white	89	E	\$10.00
76	LakukAggi	57	E	\$10.00

## To add an ActiveReport to the Visual Studio project

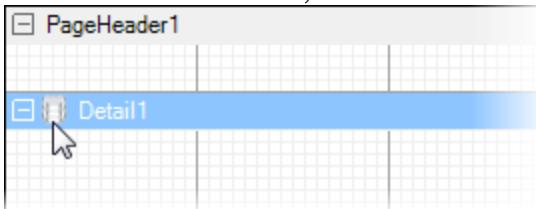
1. Create a new Visual Studio project.
2. From the **Project** menu, select **Add New Item**.
3. In the Add New Item dialog that appears, select **ActiveReports 7 Section Report (code-based)** and in the Name field, rename the file as **rptModifyDS**.
4. Click the **Add** button to open a new section report in the **designer**.

See **Adding an ActiveReport to a Project** 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 find the database path**

1. Right-click in any section of the design window of **rptModifyDS**, and select **View Code** to display the code view for the report.
2. Add code to the report to get the sample database path from the registry.

**To write the code in Visual Basic.NET****Visual Basic.NET code. Paste JUST BELOW the using GrapeCity.ActiveReports statements at the top of the code view.**

```
Imports Microsoft.Win32
```

**Visual Basic.NET code. Paste INSIDE the report class.**

```
Private Function getDatabasePath() As String
 Dim regKey As RegistryKey
 regKey = Registry.LocalMachine
 regKey = regKey.CreateSubKey("C:\Users\UserName\Documents\ComponentOne
Samples\ActiveReports Developer 7\Data\nwind.mdb")
 getDatabasePath = CType(regKey.GetValue(""), String)
End Function

}
```

**To write the code in C#****C# code. Paste JUST BELOW the using DataDynamics.ActiveReports; statements at the top of the code view.**

```
using Microsoft.Win32;
```

**C# code. Paste INSIDE the report class.**

```
private string getDatabasePath()
{
 RegistryKey regKey = Registry.LocalMachine;
 regKey = regKey.CreateSubKey("C:\\Users\\UserName\\Documents\\ComponentOne
Samples\\ActiveReports Developer 7\\Data\\nwind.mdb");
 return ((string)(regKey.GetValue(""))));
}
```

---

## 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 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 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 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

### 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 **Using the Viewer** for further information.

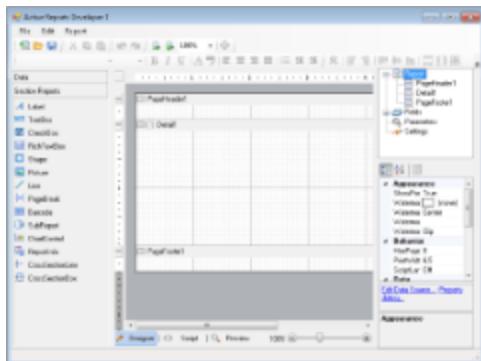
## Creating a Basic End User Report Designer (Pro Edition)

Using ActiveReports Developer Professional Edition, you can set up a custom end-user report designer. This walkthrough illustrates how to set up a basic end-user report designer on a Windows Forms application (The Designer control is not supported on a Web application).

This walkthrough is split into the following activities:

- Adding controls to the form
  - Adding code to import the toolbox library
  - Adding an OnExit method
  - Adding code to create a data toolbox group
  - Adding code to set up the toolbox, menus and toolstrips
  - Viewing the End User Report Designer

When you have finished this walkthrough, you will have a working end-user report designer that looks like the following.



 **Note:** If you need help with adding the Designer controls to your Visual Studio toolbox, see [Adding ActiveReports Controls](#).

### To add controls to the Form

1. Select the Windows Form in your application and go to the Properties Window to change the **Name** property to **formDesigner** and **Text** to **Active Reports Developer 7**.
2. Resize the Form so that you can comfortably add the controls.
3. From the Visual Studio toolbox, drag the following controls onto the Form in the order listed below, setting the properties as indicated (If you have not yet added the ActiveReports controls to your toolbox, see the [Adding ActiveReports Controls](#) topic).

#### Controls for the form

Control	Parent	Name	Dock	Property Value
ToolStripContainer	formDesigner	ToolStripContainer1	Fill	LeftToolStripPanel - Enabled = False RightToolStripPanel - Enabled = False
SplitContainer	ToolStripContainer1	SplitContainer1	Fill	FixedPanel = Panel1
Designer	SplitContainer1.Panel2	arDesigner	None	Anchor = Top, Bottom, Left, Right Resize and move as necessary.
ReportExplorer	SplitContainer1.Panel2	arReportExplorer	None	ReportDesigner = arDesigner Anchor = Top, Right Resize and move as necessary.
PropertyGrid	SplitContainer1.Panel2	arPropertyGrid	None	Anchor = Top, Bottom, Right Resize and move as necessary.
Toolbox	SplitContainer1.Panel1	arToolbox	Fill	-

4. Select arDesigner and in the Properties window, drop down the **PropertyGrid** property and select **arPropertyGrid**.
5. With the controls added in the correct order and all of the above properties set, the form looks similar to the following:



### To import the Toolbox library

1. Right-click the form and select **View Code**.
2. In the code view that appears, add the following code to give your project access to the Toolbox library.

The following examples show what the code looks like.

### To write the code in Visual Basic.NET

#### Visual Basic.NET code. Paste ABOVE the **formDesigner** class.

```
'Add the following Imports statements
Imports GrapeCity.ActiveReports
Imports GrapeCity.ActiveReports.Design
Imports GrapeCity.ActiveReports.Design.Toolbox
```

```
Imports GrapeCity.ActiveReports.SectionReportModel
Imports GrapeCity.ActiveReports.Data
```

## To write the code in C#

### C# code. Paste ABOVE the formDesigner class.

```
//Add the following using statements
using GrapeCity.ActiveReports;
using GrapeCity.ActiveReports.Design;
using GrapeCity.ActiveReports.Design.Toolbox;
using GrapeCity.ActiveReports.SectionReportModel;
using GrapeCity.ActiveReports.Data;
```

## To add the OnExit method

1. Right-click in any section of formDesigner, and select **View Code**.
2. In the code view that appears, add the following code to create an OnExit method that you can call from the Exit menu item we create later.

The following examples show what the code looks like.

## To write the code in Visual Basic.NET

### Visual Basic.NET code. Paste INSIDE the formDesigner class.

```
Private Sub OnExit(ByVal sender As Object, ByVal e As EventArgs)
 MyBase.Close()
End Sub
```

## To write the code in C#

### C# code. Paste INSIDE the formDesigner class.

```
private void OnExit(object sender, EventArgs e)
{
 Close();
}
```

## To create a data toolbox group

1. Add the following code to create a data group on the toolbox.
2. This code creates a LoadTools method that you can call in the formDesigner\_Load event to load the new toolbox group into the toolbox.

The following examples show what the code looks like.

## To write the code in Visual Basic.NET

### Visual Basic.NET code. Paste INSIDE the formDesigner class.

```
Private Sub LoadTools(ByVal arToolbox As
GrapeCity.ActiveReports.Design.Toolbox.Toolbox)
 'Add Data Providers
 Me.arToolbox.AddToolboxItem(New
System.Drawing.Design.ToolboxItem(GetType(System.Data.DataSet)), "Data")
 Me.arToolbox.AddToolboxItem(New
System.Drawing.Design.ToolboxItem(GetType(System.Data.DataView)), "Data")
 Me.arToolbox.AddToolboxItem(New
System.Drawing.Design.ToolboxItem(GetType(System.Data.OleDb.OleDbConnection)), "Data")
 Me.arToolbox.AddToolboxItem(New
System.Drawing.Design.ToolboxItem(GetType(System.Data.OleDb.OleDbDataAdapter)), "Data")
```

```
Me.arToolbox.AddToolboxItem(New
System.Drawing.Design.ToolboxItem(GetType(System.Data.Odbc.OdbcConnection)), "Data")
 Me.arToolbox.AddToolboxItem(New
System.Drawing.Design.ToolboxItem(GetType(System.Data.Odbc.OdbcDataAdapter)), "Data")
 Me.arToolbox.AddToolboxItem(New
System.Drawing.Design.ToolboxItem(GetType(System.Data.SqlClient.SqlConnection)),
"Data")
 Me.arToolbox.AddToolboxItem(New
System.Drawing.Design.ToolboxItem(GetType(System.Data.SqlClient.SqlDataAdapter)),
"Data")
End Sub
```

---

## To write the code in C#

### C# code. Paste INSIDE the formDesigner class.

```
private void LoadTools(GrapeCity.ActiveReports.Design.Toolbox.Toolbox arToolbox)
{
 //Add Data Providers
 this.arToolbox.AddToolboxItem(new
System.Drawing.Design.ToolboxItem(typeof(System.Data.DataSet)), "Data");
 this.arToolbox.AddToolboxItem(new
System.Drawing.Design.ToolboxItem(typeof(System.Data.DataView)), "Data");
 this.arToolbox.AddToolboxItem(new
System.Drawing.Design.ToolboxItem(typeof(System.Data.OleDb.OleDbConnection)), "Data");
 this.arToolbox.AddToolboxItem(new
System.Drawing.Design.ToolboxItem(typeof(System.Data.OleDb.OleDbDataAdapter)), "Data");
 this.arToolbox.AddToolboxItem(new
System.Drawing.Design.ToolboxItem(typeof(System.Data.Odbc.OdbcConnection)), "Data");
 this.arToolbox.AddToolboxItem(new
System.Drawing.Design.ToolboxItem(typeof(System.Data.Odbc.OdbcDataAdapter)), "Data");
 this.arToolbox.AddToolboxItem(new
System.Drawing.Design.ToolboxItem(typeof(System.Data.SqlClient.SqlConnection)),
"Data");
 this.arToolbox.AddToolboxItem(new
System.Drawing.Design.ToolboxItem(typeof(System.Data.SqlClient.SqlDataAdapter)),
"Data");
}
```

---

## To set up the designer's toolbox, menus and toolstrips

1. In the Design view of the form, double-click the title bar of formDesigner. This creates an event-handling method for the formDesigner Load event.
2. Add code to the handler to:
  - Set up the toolbox
  - Set up the menu and tool strips
  - Add an Exit command to the menu

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 formDesigner Load event.

```
' Add controls to the toolbox
LoadTools(arToolbox)
arDesigner.Toolbox = arToolbox

' Add Menu and ToolStrips to Form
```

```
Dim menuStrip As ToolStrip = arDesigner.CreateToolStrips(DesignerToolStrips.Menu) (0)
Dim fileMenu As ToolStripDropDownItem = CType(menuStrip.Items(0),
ToolStripDropDownItem)

' Add an Exit command to the File menu
fileMenu.DropDownItems.Add(New ToolStripMenuItem("Exit", Nothing, AddressOf OnExit))

Dim panel As ToolStripPanel = ToolStripContainer1.TopToolStripPanel
panel.Join(menuStrip, 0)
panel.Join(arDesigner.CreateToolStrips(DesignerToolStrips.Zoom) (0), 1)
panel.Join(arDesigner.CreateToolStrips(DesignerToolStrips.Undo) (0), 1)
panel.Join(arDesigner.CreateToolStrips(DesignerToolStrips.Edit) (0), 1)
panel.Join(arDesigner.CreateToolStrips(DesignerToolStrips.Report) (0), 1)

panel.Join(arDesigner.CreateToolStrips(DesignerToolStrips.Layout) (0), 2)
panel.Join(arDesigner.CreateToolStrips(DesignerToolStrips.Format) (0), 2)
```

---

## To write the code in C#

### C# code. Paste INSIDE the formDesigner Load event.

```
// Add controls to the toolbox
LoadTools(arToolbox);
arDesigner.Toolbox = arToolbox;

// Add Menu and CommandBar to Form
ToolStrip menuStrip = arDesigner.CreateToolStrips(DesignerToolStrips.Menu) [0];

ToolStripDropDownItem fileMenu = (ToolStripDropDownItem)menuStrip.Items[0];

// Add an Exit command to the File menu
fileMenu.DropDownItems.Add(new ToolStripMenuItem("Exit", null, this.OnExit));
ToolStripPanel panel = ToolStripContainer1.TopToolStripPanel;
panel.Join(menuStrip, 0);
panel.Join(arDesigner.CreateToolStrips(DesignerToolStrips.Zoom) [0], 1);
panel.Join(arDesigner.CreateToolStrips(DesignerToolStrips.Undo) [0], 1);
panel.Join(arDesigner.CreateToolStrips(DesignerToolStrips.Edit) [0], 1);
panel.Join(arDesigner.CreateToolStrips(DesignerToolStrips.Report) [0], 1);

panel.Join(arDesigner.CreateToolStrips(DesignerToolStrips.Layout) [0], 2);
panel.Join(arDesigner.CreateToolStrips(DesignerToolStrips.Format) [0], 2);
```

---

## To view an End User Report Designer

1. Press F5 to run the project. The **End User Report Designer** opens with a section layout.
2. Go to the **File** menu and select **New** to open the Create New Report dialog.
3. From **Templates**, select **Page Report** to open a page layout in the designer along with its corresponding toolbox controls and report explorer items.
4. Go to the **File** menu again. Notice that an **Exit** menu item has been added to the listed menu items.

## Customizing the Flash Viewer UI

You can customize the Flash Viewer UI, using javascript functions. This walkthrough illustrates how to customize the look and feel of the Flash Viewer.

The walkthrough is split up into the following activities:

- Adding code to customize the Flash Viewer options
- Adding code to handle the Flash Viewer events

To follow the steps of this walkthrough, you must first complete the steps on creating an ASP.NET Web site and setting up the FlashViewer, described in the Flash Viewer walkthrough.

## To customize the Flash Viewer options

1. Select the WebViewer on your .aspx page, and in the Properties window, expand the **FlashViewerOptions** node.
2. Set the property **UseClientApi** (FlashViewerOptions) to True (this property is set to False by default).
3. Declare a variable and attach the FlashViewer by adding the following code onto the .aspx source view.

### Paste the code into .aspx source

```
<script language="javascript" type="text/javascript">
var viewer;

function init() { GrapeCity.ActiveReports.Viewer.OnLoad ("WebViewer1", function()
{
 viewer = GrapeCity.ActiveReports.Viewer.Attach("WebViewer1");
});
</script>

...
<body onload="return init()">
```

4. Drag the **Html Input (button)** control from the Visual Studio toolbox onto the .aspx design view, containing the WebViewer control.
5. Double-click the **Button** control and paste the following code:

### Paste the code into .aspx source

```
<input id="button1" type="button" value="button" onclick="return
Button1_onclick()" />
```

6. Add the following function for this button:

### Paste the code into .aspx source

```
function Button1_onclick() {
 var zoom = viewer.getZoom();
 alert("Check that zoom property is changed from " + zoom);
 zoom += 0.1;
 viewer.setZoom(zoom);
 alert("to " + zoom);
}
```

 **Note:** You can declare a variable and attach the FlashViewer in the event handler directly.

### Paste the code into .aspx source

```
function Button1_onclick()
{
var viewer;
viewer = GrapeCity.ActiveReports.Viewer.Attach("WebViewer1");
...
}
```

}

## Available Flash Viewer properties

 The properties below use a pair of the getter and setter functions, which names are formed as {get|set} <PropertyName>().

Property Name	Description
<b>CurrentPage</b>	Integer. Gets or sets the page for a currently selected view.
<b>HyperLinkBackColor</b>	String. Gets or sets the background color for all hyperlinks in a document. This value is applied only after a new document is loaded. The format is "#FFoooo" (#RRGGBB).
<b>HyperLinkForeColor</b>	String. Gets or sets the color for all hyperlinks in a document. This value is applied only after a new document is loaded. The format is "#FFoooo" (#RRGGBB).
<b>HyperLinkUnderline</b>	Boolean. Gets or sets a value determining whether the text of all hyperlinks in a document is underlined. This value is applied only after a new document is loaded.
<b>SearchResultsBackColor</b>	String. Gets or sets the background color of the highlighted text when using the Find dialog. The format is "#FFoooo" (#RRGGBB).
<b>SearchResultsForeColor</b>	String. Gets or sets the color of the highlighted text when using the Find dialog. The format is "#FFoooo" (#RRGGBB).
<b>PaperColor</b>	String. Gets or sets the paper color of the report. The format is "#FFoooo" (#RRGGBB).
<b>ShowSplitter</b>	Boolean. Gets or sets a value determining whether to display a splitter. If set to True, the splitter is shown.
<b>TargetView</b>	String. Gets or sets a value that specifies which view - Primary or Secondary, is currently active. If the <b>ShowSplitter</b> property is set to False, then you cannot switch to the Secondary view.
<b>ThemeUrl</b>	String. Gets or sets a value specifying the relative URL of a skin to use on the Flash Viewer.
<b>Zoom</b>	Integer. Gets or sets a value that specifies the zoom level at which to display the report.
<b>TocPanel</b>	Object. Gets the Table of Contents Panel object. This property is read-only. <ul style="list-style-type: none"> <li>• <b>RightAlign</b> To align the TOC panel to the right of the Viewer.</li> <li>• <b>ThumbnailsVisible</b> To specify whether to display a pane with thumbnail views of pages.</li> <li>• <b>TocVisible</b> To specify whether to display the table of contents pane.</li> <li>• <b>Visible</b> To specify whether to display the Table of contents.</li> <li>• <b>Width</b> To specify the Table of contents width.</li> </ul>
<b>ViewType</b>	String. Gets or sets the current ViewType value.
<b>EventsHandler</b>	Object. Gets or sets the events handler container.

## Available Flash Viewer methods

Method Name	Description
<b>LoadDocument(string documentUrl)</b>	Loads the RDF document. You may use an RDF file that resides on a server, RpxHandler, CompiledReportHandler or custom Http handler implementation to provide a streamed document. This method cancels the current document's

	loading.
<b>CancelDocumentLoad()</b>	Cancels the loading of a current document.
<b>Print(PrintOptions options)</b>	Causes the Viewer to wait until all required pages are loaded, displayed in the Print dialog and starts printing. The <b>PrintOptions</b> are similar to the WebViewer.PrintOptions, except that the PageRangesCollection methods are merged into the PrintOptions class.
<b>CreatePrintOptions()</b>	Creates options for the <b>Print()</b> method.
<b>setViewType(string viewType, int multiPageRows, int multiPageCols)</b>	Specifies the view mode. Possible values for the first parameter are specified in the <b>ViewType</b> enumeration. The last two parameters are applied for <b>ViewType.MultiPage</b> only.

## To handle Flash Viewer Events

1. Declare a variable, attach the Flash Viewer and add event handlers by adding the following code onto the .aspx source view.

### Paste the code into .aspx source

```
<script language="javascript" type="text/javascript">
var viewer;
function init() {
GrapeCity.ActiveReports.Viewer.OnLoad("WebViewer1", function() {
viewer = GrapeCity.ActiveReports.Viewer.Attach("WebViewer1");
 viewer.setEventsHandler({
 OnLinkClick: function(e) {
 alert(e.Link); //specifies url of the link item, string
 return true;
 },
 OnError: function(e) {
 alert(e.Message); //error message, string
 alert(e.ErrorType); //possible types are "Error" and "Warning", string
 return false;
 }
 });
})
}
</script>
...
<body onload="return init()">
```

 **Note:** <EVENTS> are described in detail in the **Available events** list below.

### Available events

Event	Description
<b>OnLinkClick(LinkEventArgs)</b>	Specifies the URL value of a linked item or a string. This event is raised when a report object with a hyperlink is clicked; this event overrides the default Hyperlinks behavior that simply opens another

browser window. Cancelable.

The event handler receives an argument of type LinkEventArgs containing data related to this event. The handler argument has the field "Link". The field "Link" returns the hyperlink URL value.

#### Code example

```
OnLinkClick: function(e)
{
 alert(e.Link); //specifies url of the
 link item, string
 return true;
}
```

---

#### OnError(ErrorEventArgs)

Fires when an application fires an error or a warning. Use this event to customize FlashViewer error messages and warnings.

The event handler receives an argument of type ErrorEventArgs containing data related to this event. The handler argument has the following fields - "ErrorType" and "Message".

The field "ErrorType" contains the error type value; the field "Message" contains a description of a Flash Viewer problem.

This event allows suppressing any notifications to a user.

#### Code example

```
OnError: function(e)
{
 alert(e.Message);
 //error message, string
 alert(e.ErrorType);
 //possible types are "Error" and
 "Warning", string
 return false;
}
```

---

#### OnLoadProgress(LoadProgressArgs)

Raised during the processing of a report.

The event handler receives an argument of type LoadProgressArgs containing data related to this event. The handler argument has the following fields - "PageCount", "PageNumber" and "State".

The field "PageCount" returns the total count of report pages.

The field "PageNumber" returns the page number of the processed report.

The field "State" returns the value, indicating the state of the report's processing; the possible values are "Completed", "InProgress" and "Cancelled".

**Code example**

```
OnLoadProgress: function(e)
{
 if(e.State == "InProgress") {
 if(e.PageNumber == 10)
 {
 alert("10 pages
are loaded");
 }
}

if(e.State == "Cancelled") {
 alert("Report
processing is cancelled");
}

if(e.State == "Completed") ///
possible value are Completed,
InProgress and Cancelled
{
 alert("Report loading
is completed, total page count is" +
e.PageCount);
}

//return value is ignored in
this event
}
```

---

**OnTargetViewChanging(TargetViewChangeEventArgs)**

Raised while a report's view is changing.

The event handler receives an argument of type TargetViewChangedEventArgs containing data related to this event. The handler argument has the following fields - "CurrentView" and "NewView".

The field "CurrentView" returns the currently selected view value.

The field "NewView" returns the newly selected view value.

**Code example**

```
OnTargetViewChanging: function(e)
{
 alert("Currently
selected view is " + e.CurrentView);
//gets currently selected view, string
 alert("Newly selected
view is " + e.NewView);
//gets newly selected view, string
 return false;
//cancelable event
}
```

---

**OnToolClick(ToolEventArgs)**

Raised by clicking the toolbar button. Cancelable.

The event handler receives an argument of type ToolEventArgs containing data related to this event. The handler argument has the following field - "Tool".

The field "Tool" returns the name of a clicked button.

**Code example**

```
OnToolClick: function(e)
{
 alert(e.Tool);
 return false;
}
```

**OnCurrentPageChanged(CurrentPageChangedEventArgs)**

Raised each time a current page is changed within the current view programmatically or by any user navigation command. This event is also raised when the current view (primary or secondary) is changed.

The event handler receives an argument of type CurrentPageChangedEventArgs containing data related to this event. The handler argument has the following fields - "PageNumber" and "ViaApi".

The field "PageNumber" contains the 1-based page number. The field "ViaApi" specifies whether the event is raised by setting the page number in the **CurrentPage** property.

 **Note:** Use the "return true;" value to show that the client side has handled the event. The "return false;" value indicates that the event was not handled.

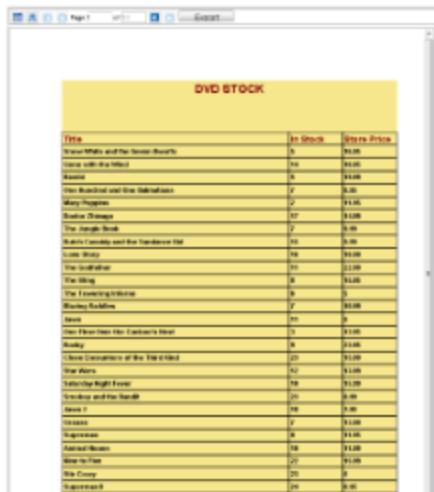
## Customizing the HTML Viewer UI

You can customize the HTMLViewer interface using JQuery methods. WebViewer control adds JQuery library in page scripts. Use the code in this walkthrough to remove a button from the HTMLViewer toolbar and add a client side PDF export implementation.

This walkthrough is split into the following activities:

- Loading an ActiveReport to a Visual Studio ASP.NET Web application.
- Accessing WebViewer's view model.
- Removing an existing toolbar button.
- Adding a function for PDF exporting.
- Binding the Custom UI to WebViewer's view model.

When you complete this walkthrough you get an HTMLViewer that looks similar to the following at runtime.



## To load an ActiveReport in the Web application

1. Create a new Visual Studio ASP.NET Web application.
  2. In the Default.aspx that gets created, go to the Visual Studio toolbox **ActiveReports 7 Page Report** tab and drag a WebViewer control onto the Design view.
  3. Load the report generated at the end of the **Single Layout Reports** walkthrough.

**Note:** You may load any report, section or page in the HTMLViewer. See [Getting Started with the Web Viewer](#) for information on loading a report.

## To access the WebViewer view model

The HTML Web Viewer 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 [Working with HTML Viewer using Javascript](#) for more information.

Follow the steps below to access the ViewModel.

1. In the Source view of the **Default.aspx** file, add a <script> tag.

**Add a script tag like the following before the body tag**

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

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 language="javascript" type="text/javascript">
function viewer_loaded()
{
};
function document_onload()
{
};
</script>
...
<body onload="return 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**

```
var viewModel = GetViewModel('WebViewer1');
```

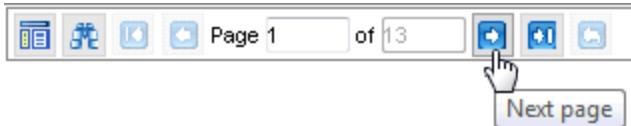
4. Add the following Javascript code inside the **document\_load** event handler to bind WebViewer's Loaded event to client side viewer\_loaded event:

**Paste the code into .aspx source**

```
$('#WebViewer1').bind('loaded', viewer_loaded);
```

## To remove a button from the WebViewer toolbar

In the Source view of the Default.aspx file, add the following Javascript code inside the **viewer\_loaded** event handler to access the WebViewer toolbar and remove the **Next Page** button from the toolbar:



**Paste the code into .aspx source**

```
var toolbar = $('#WebViewer1').find('.arvToolBar');
toolbar.find('.btnNext').remove();
```

**Note:** In order to access the child elements of the WebViewer toolbar you need their css class names. Following is a list for reference:

### Toolbar Child Elements

**Toggle Sidebar button:** btnToggleSidebar  
**Find button:** btnFind  
**First page button:** btnFirst  
**Previous page button:** btnPrev  
**Page label:** toolbarLabel  
**Page number input box:** toolbarInput  
**Next page button:** btnNext  
**Last page button:** btnLast  
**Back to parent report button:** btnBack

## To add a function for exporting to PDF

The steps below illustrate how to build a custom UI.

1. In the Source view of the Default.aspx file, add following Javascript code inside <script> tag to create a function. This function accesses the view model and exports the report in PDF format :

**Paste the code into .aspx source**

```
function exportPDF()
{

 var viewModel = GetViewModel('WebViewer1');

 if (viewModel.PageLoaded())
 {
 viewModel.Export(ExportType.Pdf, function (uri)
 {
 window.location = uri;
 }, true);
 }
}
```

```
 }
}
```

- 
2. In the Source view of the Default.aspx file, add the following Javascript code inside <form> tag to add a button on a custom toolbar. This button is enabled once the report is loaded and calls the exportPDF() function at runtime:

#### Paste the code into .aspx source

```
<div id="customToolbar" style = "display:inline">
 <button data-bind='enable: PageLoaded, click: exportPDF' style=" width: 105px;
 font-size: medium; height: 22px;">Export</button>
</div>
```

- 
3. In the Source view of the Default.aspx file, add the following Javascript code inside the **viewer\_loaded** event handler to attach the custom toolbar with the built-in toolbar:

#### Paste the code into .aspx source

```
$('#customToolbar').appendTo(toolbar);
```

---

### To bind the custom UI to the WebViewer view model and run the application

1. Add the following Javascript code inside the **viewer\_loaded** event handler to bind the custom UI to WebViewer view model:

#### Paste the code into .aspx source

```
ko.applyBindings(viewModel, document.getElementById("customToolbar"));
```

- 
2. Press F5 to run the application.

 **Note:** Replace 'WebViewer' in the code snippets above, with the actual ID of the WebViewer control in your application.

## Web Services

ActiveReports Developer provides support for web services to be used to return a dataset as a data source for a report or to return an ActiveReport document to show in a Windows Forms viewer. The following walkthroughs show how to create a simple web service for each scenario and how to create a Windows client application for each web service.

### **DataSet Web Service**

Describes how to set up a simple web service that returns a dataset.

### **DataSet Windows Application**

Describes how to set up a Windows client application for the dataset Web Service.

### **Document Web Service**

Describes how to set up a simple web service that returns an ActiveReports document.

### **Document Windows Application**

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.

## 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 Developer.

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 to the project

### To add a reference to a web service in Visual Studio 2008 or 2010 that is compatible with .NET Framework 2.0 Web service

1. From the **Project** menu, select **Add Service Reference**.
2. In the **Add Service Reference** window that appears, click **Advanced** button.
3. In the **Service Reference Settings** window that appears, click **Add Web Reference** button.
4. In the **Add Web Reference** window that appears, click the **Web services on the local machine** link.
5. Click the link to 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.)
6. Click the **Go** button, and then click the **Add Reference** button when the Web Service is recognized.

### To add a reference to a web service in Visual Studio 2008 or 2010

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 2008 or 2010 compatible with .NET Framework 2.0 Web service)

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
MeDataMember = "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;
thisDataMember = "Products";
```

---

## To set the report data source (for Visual Studio 2008 or 2010)

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 (Visual Studio 2008 or 2010 project)

 **Note:** You need to update the app.config file if you added the Service Reference to the Visual Studio 2008 or 2010 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 **maxLength** to some large number, for example, 60500.

## Document Web Service

With ASP.NET and ActiveReports Developer, 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 Developer 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 **Single Layout Reports** (for a page report) or **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 Developer 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.

#### To write the code in Visual Basic.NET

The following example demonstrates how you create the Web Method for a page 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 GrapeCity.ActiveReports.PageReport(New
System.IO.FileInfo(Server.MapPath("") + "\PageReport.rdlx"))
 rpt.Run()
 Dim rdfe As New GrapeCity.ActiveReports.Export.Rdf.RdfRenderingExtension
 Dim ms As New GrapeCity.ActiveReports.Rendering.IO.MemoryStreamProvider
 rpt.Document.Render(rdfe, ms)
 Dim doc_rdf As New GrapeCity.ActiveReports.Document.SectionDocument
 doc_rdf.Load(CType(ms.GetPrimaryStream.OpenStream, System.IO.MemoryStream))
 Return doc_rdf.Content
End Function
```

The following example 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
```

#### To write the code in C#

The following example demonstrates how you create the Web Method for a page report.

#### C# code. REPLACE the existing WebMethod and function with this code.

```
<WebMethod(
 Description:="Returns a products report grouped by category")> _
Public Function GetProductsReport() As Byte()
{
 rpt As New GrapeCity.ActiveReports.PageReport(New
System.IO.FileInfo(Server.MapPath("") + "\PageReport.rdlx"));
 rpt.Run();
 rdfe As New GrapeCity.ActiveReports.Export.Rdf.RdfRenderingExtension;
 ms As New GrapeCity.ActiveReports.Rendering.IO.MemoryStreamProvider
 rpt.Document.Render(rdfe, ms);
```

```
doc_rdf As New GrapeCity.ActiveReports.Document.SectionDocument;
doc_rdf.Load(CType(ms.GetPrimaryStream.OpenStream, System.IO.MemoryStream));
return doc_rdf.Content;
}
```

The following example demonstrates how you create the Web Method for a section report.

### 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 localhost 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 Developer, you can use a Web Service that returns the content of an ActiveReport to show in the Windows Forms viewer control.

This walkthrough illustrates how to create a Windows client application that returns the content of an ActiveReport Developer 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

## 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 2008 or 2010 that is compatible with .NET Framework 2.0 Web service

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 2008 or Visual Studio 2010

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 (for Visual Studio 2008 or 2010 compatible with .NET Framework 2.0 Web service)

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 (for Visual Studio 2008 or 2010)

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 the Visual Studio 2008 or 2010 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 **maxLength** to some large number, for example, 60500.

## To run the project

Press **F5** to run the project.

# DataSet Web Service

With ASP.NET, you can set up a Web Service that returns a dataset to use in ActiveReport Developer. 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 Service Application**.
3. Change the name of the project.
4. Click **OK** to open the new project in Visual Studio.

## To create the Web Method

In the App\_Code/Service.vb or Service.cs file that displays by default, 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=C:\Users\[User Name]\Documents\ComponentOne Samples\ActiveReports Developer
7\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 = C:\\\\Users\\\\[User Name]\\\\Documents\\\\ComponentOne Samples\\\\ActiveReports
Developer 7\\\\Data\\\\nwind.mdb";
[WebMethod>Description="Returns a DataSet containing all Products"]
public Data.DataSet GetProduct()
{
 Data.OleDb.OleDbDataAdapter adapter;
 Data.DataSet ds;
 adapter = new Data.OleDb.OleDbDataAdapter("select * from products", connString);
 ds = new 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, enter localhost 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 DataSet Web Service in an ActiveReport, see **DataSet Windows Application**.

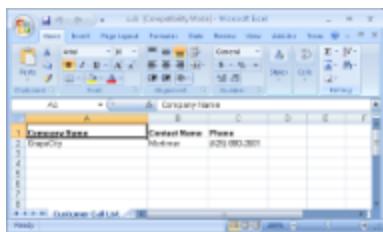
## Basic Spreadsheet with SpreadBuilder

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.v7 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 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 an ActiveReport layout template and in the Name field, rename the file as **BasicSpreadsheet**.
4. Click the **Add** button to open a new section report in the **designer**.

See **Adding an ActiveReport to a Project** for information on adding different report layouts.

### To add an GrapeCity.ActiveReports.Export.Excel.v7 assembly reference to your project

1. From the Visual Studio **Project** menu, select **Add Reference**.
2. In the Add Reference window that appears, select GrapeCity.ActiveReports.Export.Excel.v7 assembly reference and click **OK**.

**Note:** In ActiveReports Developer, by default, the assemblies are located in the ...\\Common Files\\ComponentOne\\ActiveReports Developer 7 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.

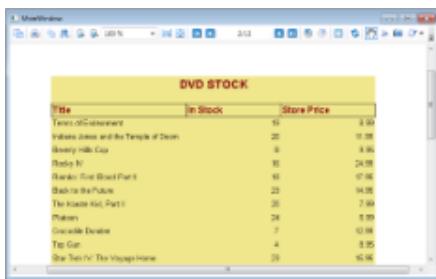
## WPF Viewer

The ActiveReports WPF Viewer is a custom control that allows to easily view both section and page report layouts.

This walkthrough is split up into the following activities.

- Creating a WPF Application project in Visual Studio 2010
- Adding the ActiveReports WPF Viewer control to the xaml page
- Binding a report to the ActiveReports WPF Viewer
- Previewing a report
- Customizing the ActiveReports WPF Viewer

When you have completed this walkthrough, you will have the ActiveReports WPF Viewer displaying a report that looks similar to the following.



### To create a WPF application in Visual Studio 2010

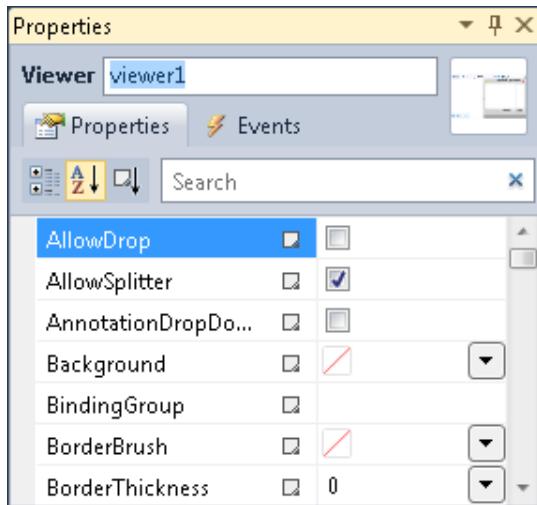
1. On the Visual Studio **File** menu, click **New Project**.
2. In the **New Project** dialog that appears, select **WPF Application** in the list of templates.
3. Specify the name and location of the project and then click the **OK** button.
4. In the Visual Studio Solution Explorer, right-click *YourProject* and select **Add**, then **New Item**.
5. In the **Add New Item** dialog that appears, select the **ActiveReports 7 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 7 tab, drag the **Viewer** control and drop it on the design view of *MainWindow.xaml*.
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 bind 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**.
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");
```

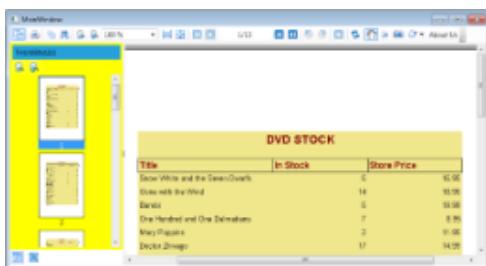
**Note:** For an example of other ways to bind a report to the WPF Viewer, see the **LoadDocument ('LoadDocument Method' in the on-line documentation)** method in the Class Library documentation.

#### 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 (DefaultWPFViewerTemplates.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\\ComponentOne\\ActiveReports Developer 7\\Deployment\\WPF\\Templates folder (on a **64-bit Windows operating system**, this file is located in [*systemdrive*]\\Program Files (x86)\\ComponentOne\\ActiveReports Developer 7\\Deployment\\WPF\\Templates).
5. On MainWindow.xaml before the opening <Grid> tag, add the following code.

### Paste to the Design 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={StaticResource res}, Path=Resources.ThumbnailsPanel\_Title}" and type "**THUMBNAILS**".
4. Search for "**TabControl x:Name="Sidebar"**".
5. In the Properties Window, go to the **Background** property and select the color "**Yellow**".
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;
```

```
 }

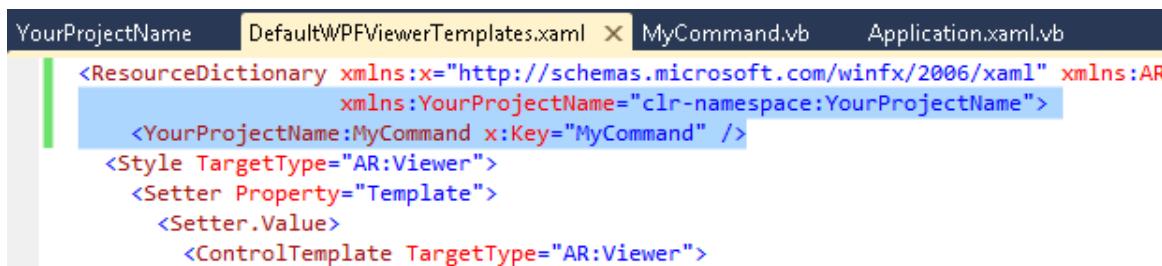
 public void Execute(object parameter)
 {
 MessageBox.Show("GrapeCity is the world's largest component vendor.", "About Us", MessageBoxButtons.OK);
 }

 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>
...
<YourProjectName: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.

## 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

### References missing from Visual Studio Add Reference dialog

**Symptoms:** When you try to add references to your project, only a few of the ActiveReports version 7 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 4.0.

## 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.Interop64.v7.dll is not available in the default "Add Reference" dialog

**Symptoms:** The GrapeCity.ActiveReports.Interop64.v7.dll is not available in the default "Add Reference" dialog.

**Cause:** The GrapeCity.ActiveReports.Interop64.v7.dll is located in the distribution folder.

**Solution:** The GrapeCity.ActiveReports.Interop64.v7.dll is located in **C:\Program Files (x86)\Common Files\ComponentOne\ActiveReports Developer 7\redist**.

## 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 7 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:** There are two ways to resolve this issue. One is to set the **UseEvidence** property to False.

The other is to update the configuration file. 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>
```

## Microsoft Access OLE DB provider in a 64-bit system

**Symptoms:** Microsoft Access OLE DB provider, Microsoft.Jet.OLEDB.4.0 does not work on a 64-bit system.

**Cause:** In Visual Studio 2008, by default, projects are set to use 32 bit or 64 bit, depending on the environment on which they are run. The Microsoft Access OLE DB provider, Microsoft.Jet.OLEDB.4.0, is not compatible with 64 bit, so it fails with Visual Studio 2008 on a 64-bit system.

**Solution:** To avoid this situation, 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 both Page reports and Section reports.

1. In the project where you call the Print method, add a reference to the GrapeCity.ActiveReports.Viewer.Win.v7 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.

## ActiveReports controls do not appear in the toolbox

**Symptoms:** ActiveReport controls do not appear in the toolbox even when they are added manually using the steps in **Adding ActiveReports Controls**.

**Cause:** The project is using .NET 2.0 or lower.

**Solution:** Confirm if the project is using .NET 3.5 or later. .NET 2.0 is not supported in ActiveReports Developer.

## 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 4.0 Client Profile or .NET Framework 4.0 Full Profile 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.

## Page 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 a CPL report

**Symptoms:** Cannot set a dataset field (bound field) in the PageHeader of a CPL report.

**Cause:** ActiveReports Developer is based on the RDL 2005 specifications, therefore, referencing datasets in the PageHeader of a CPL 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 an FPL report instead, 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.

**Cause:** Document property is available for section reports only.

**Cannot add assembly reference created in .NET Framework 4.0 or above in PageReports**

**Symptoms:** Cannot add assembly reference created in .NET Framework 4.0 or above in PageReports of the Standalone Designer application.

**Cause:** Standalone Designer application has been created in .NET 3.5 framework, therefore it cannot load .NET 4.0 assemblies.

## Flash Viewer Troubleshooting

### Swfobject undefined error

**Symptoms:** With the Flash viewer, my page throws a swfobject is undefined error.

**Cause:** The ActiveReports Handler Mappings are not set up correctly in IIS 7.0.

**Solution: Configure HTTPHandlers in IIS 7.x.**

### IOError while loading document. Error #2032

**Symptoms:** When running the Flash viewer in IIS, an error occurs with the following message: "IOError while loading document. Reason: Error #2032."

**Cause:** The ActiveReports Handler Mappings are not set up correctly in IIS or in the web.config file.

**Solution:** Update **Configure HTTPHandlers in IIS 7.x** or if that is already done, ensure that the handlers are enabled in your web.config file.

### FireFox displays white pages

**Symptoms:** When running the Flash viewer in FireFox, reports display white pages, but Internet Explorer renders reports correctly.

**Cause:** The height and width of the Flash viewer control is set to 100%. FireFox does not support this setting, so it does not resize the Flash Viewer at all.

**Solution:** Use cascading style sheets (CSS) to set the properties.

### To use CSS in your Flash viewer ASPX

#### ASPX CSS code

```
<style type="text/css">
html, body, #WebViewer1, #WebViewer1_controlDiv
{
 width: 100%;
 height: 100%;
 margin: 0;
}
</style>
```

---

### To use an external CSS file

1. Assign the Flash viewer control a CSS class of **report-viewer**.
2. Add code like the following to the CSS file.

#### ASPX CSS code. Paste in the external CSS file

```
.report-viewer, .report-viewer div, .report-viewer object {height: 100%; width:
100%;}
```

---

## Silverlight Viewer Troubleshooting

### JScript error while using the Silverlight Viewer in Silverlight 5

**Symptoms:** When using the Silverlight Viewer in Silverlight 5, the JScript error may occur with the following message: "Unhandled Error in Silverlight Application The invocation of the constructor on type 'DataDynamics.ActiveReports.Viewer'

that matches the specified binding constraints threw an exception."

**Cause:** The Silverlight Viewer is based on Silverlight 4, and it adds reference to System.Windows.Controls.dll (2.0.5.0). In Silverlight 5, adding the Silverlight Viewer control does not automatically add reference to System.Windows.Controls.dll to the project because the Silverlight Viewer adds reference to the Silverlight 4 version.

**Solution:** Add a reference to System.Windows.Controls.dll (in Silverlight 4 SDK or Silverlight 5 SDK) to the project.

#### Pan Mode does not work if the Silverlight viewer is placed in layout panels

**Symptoms:** Pan Mode does not work in horizontal/vertical directions, if the Silverlight viewer is placed in layout panels such as StackPanel, Canvas etc.

**Cause:** The default size of the Silverlight viewer is set to infinite according to the layout panel properties.

**Solution:** Go to the properties window and set a custom value in the Height and Width property of the layout panel.

## 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 *incorrect* 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.

## WebViewer Troubleshooting

### The WebViewer will not print without displaying the report

**Symptoms:** The WebViewer will not automatically print a report without displaying it.

**Cause:** Only the new FlashViewer ViewerType of the WebViewer offers this functionality.

**Solution:**

1. Set the **ViewerType** property to **FlashViewer**.
2. Expand the **FlashViewerOptions** property, and expand the **PrintOptions** subproperty.
3. Under the **PrintOptions** subproperty, set the **StartPrint** property to **True**.

### The report is not getting updated with new data, or the page number stays the same

**Symptoms:** The WebViewer stays on the page number last viewed in the previous report when a user selects a new report or refreshes the current report, or new data does not display on refresh.

**Cause:** If the control is loaded in response to a client postback, the Report property does not run the specified report.

Instead it uses a previously cached copy of the report's Document in the WebCache service to supply speedy responses to clients.

**Solution:** To force the client to use a new instance, call the **ClearCachedReport** method before setting the Report property.

### 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\v2.0.50727 or v3.0 or v3.5.
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**.

## Help Troubleshooting

### ActiveReports 7 Help is not updated in Visual Studio 2010 after the new service pack installation

**Symptoms:** ActiveReports 7 Help is not updated in Visual Studio 2010 after the new service pack installation.

## Solution:

1. Open the **Help Library Manager** by selecting **Manage Help Settings** in the **Microsoft Visual Studio 2010/Visual Studio Tools** folder, or in the **Help** menu of Visual Studio 2010.
2. Select **Remove content** in the **Help Library Manager**.
3. Click the **Remove** action next to the content title and then click **Remove** to remove the old ActiveReports 7 Help content.
4. Install ActiveReports 7 Help as described in **Installing Help in Visual Studio 2010**.

## Pressing the F1 key does not open ActiveReports 7 Help in Visual Studio 2010

**Symptoms:** I have installed ActiveReports 7 Help as described in **Installing Help in Visual Studio 2010** but pressing the F1 key does not open the documentation file in Visual Studio 2010.

**Solution:** Make sure that you have restarted the Help Library Agent. If restarting the Help Library Agent does not resolve this issue, please contact our support team: [powersupport@grapecity.com](mailto:powersupport@grapecity.com).

## Class Library

The Class Library contains documentation and code samples for the entire ActiveReports API.

## This section contains information about

### **GrapeCity.ActiveReports.v7 Assembly (on-line documentation)**

This assembly contains the namespaces ActiveReports, Data, Expressions, PageReportModel, and SectionReportModel.

### **GrapeCity.ActiveReports.Chart.v7 Assembly (on-line documentation)**

This assembly contains the namespaces Chart (including all axes, data, data points, legends, series, titles, and other configurable items in the charts), Annotations, Graphics, Styling, and Wizard.

### **GrapeCity.ActiveReports.Design.Win.v7 Assembly (on-line documentation)**

This assembly contains the Design namespace (including all End User Report Designer features).

### **GrapeCity.ActiveReports.Document.v7 Assembly (on-line documentation)**

This assembly contains the namespaces ActiveReports (border classes), Document.Section (including canvas drawing classes, bookmarks, paper sizes, etc.), and Export.Html (for HTML output types).

### **GrapeCity.ActiveReports.Export.Excel.v7 Assembly (on-line documentation)**

This assembly contains the Export.Excel namespace (Excel export class with export methods and properties).

### **GrapeCity.ActiveReports.Export.Html.v7 Assembly (on-line documentation)**

This assembly contains the Export.Html namespace (HTML export class with export methods and properties).

### **GrapeCity.ActiveReports.Export.Image.v7 Assembly (on-line documentation)**

This assembly contains the Export.Image namespace (Image export class with export methods and properties).

### **GrapeCity.ActiveReports.Export.Pdf.v7 Assembly (on-line documentation)**

This assembly contains the Export.Pdf namespace (PDF export, document options, and security classes).

### **GrapeCity.ActiveReports.Export.Word.v7 Assembly (on-line documentation)**

This assembly contains the Export.Word namespace (Word export class with export methods and properties).

### **GrapeCity.ActiveReports.Export.Xml.v7 Assembly (on-line documentation)**

This assembly contains the Export.Xml namespace (XML export class with export methods and properties).

### **GrapeCity.ActiveReports.Extensibility.v7 Assembly (on-line documentation)**

This assembly contains the Extensibility namespace (ResourceLocator class and interfaces used with the page report designer).

### **GrapeCity.ActiveReports.Viewer.Silverlight.v7 Assembly (on-line documentation)**

This assembly contains the namespaces ActiveReports (including the Silverlight Viewer), and ViewModel.

### **GrapeCity.ActiveReports.Viewer.Win.v7 Assembly (on-line documentation)**

This assembly contains the Windows.Forms namespace (Viewer class with methods and properties).

**GrapeCity.ActiveReports.Viewer.Wpf Assembly ('GrapeCity.ActiveReports.Viewer.Wpf.v7 Assembly' in the on-line documentation)**

This assembly contains Viewer.Wpf namespace (classes similar to Viewer.Win namespace).

**GrapeCity.ActiveReports.Web.v7 Assembly (on-line documentation)**

This assembly contains the namespaces Web (including the WebViewer), ExportOptions, and Handlers.

## Index

.NET Framework Client and Full Profile Versions, 31  
3D Pie Chart, 639-641  
ActiveReports 7 with MVC Sample, 504-506  
ActiveReports and the Web , 55  
ActiveReports Designer, 74-76  
ActiveReports Developer 7, 13  
ActiveReports Developer Guide, 13  
ActiveReports Editions, 17-23  
Add a Cascading Parameter, 331-333  
Add a Dataset, 302-304  
Add and Save Annotations, 391-393  
Add Bookmarks, 393-395 , 337-339  
Add Code to Layouts Using Script, 403-408  
Add Field Expressions, 371-373  
Add Grouping in Section Reports, 368-369  
Add Hyperlinks, 395-398 , 337  
Add Items to the Document Map, 351-353  
Add Page Breaks in CPL, 354  
Add Page Numbering, 196-197  
Add Parameters in a Page Report, 329-330  
Add Parameters in a Section Report, 389-391  
Add Static Rows and Columns to a Matrix, 359-360  
Add Totals and Subtotals in a Data Region , 354-359  
Adding a Data Source to a Report, 47  
Adding ActiveReports Controls, 45-46  
Adding an ActiveReport to a Project, 46-47  
Adding an ActiveReports Application, 72-73  
Address Labels, 676-678  
Allow Users to Sort Data in the Viewer, 347-348  
Annotations, 288-289  
Annual Report Sample, 464-465  
API, 440 , 445  
BandedList Reports, 534-538  
BandedList, 108-112  
Bar Chart, 637-639  
Barcode (Section Report) , 240-246  
Barcode, 112-117  
Basic Data Bound Reports, 626-628  
Basic Spreadsheet with SpreadBuilder, 735-737  
Basic XML-Based Reports (RPX), 657-661  
Bind a Page Report to a Data Source at Run Time, 304-313

**Bind Reports to a Data Source, 363-368**  
**Bound Data Sample, 457-458**  
**Breaking Changes, 39-42**  
**Bullet, 117-120**  
**CacheToDisk and Resource Storage, 267**  
**Calculated Fields Sample, 488-489**  
**Calendar, 120-123**  
**Category Selection Sample, 465-466**  
**Cell Merging In Matrix, 360-361**  
**Change Page Size, 353-354**  
**Change Ruler Measurements, 385-386**  
**Chart , 123-131**  
**Chart Data Dialog, 131-138**  
**Chart Walkthroughs, 637**  
**ChartControl, 248-249**  
**Charting Sample, 466-467**  
**Charts in a Page Report, 566-570**  
**CheckBox (Page Report), 138-140**  
**CheckBox (Section Report), 232-233**  
**Class Library, 750-751**  
**Code-Based Section Report, 105-106**  
**Collate Multiple Copies of a Report, 520-523**  
**Color Scale 2, 215-218**  
**Color Scale 3, 218-221**  
**Columnar Layout Reports (CPL), 531-534**  
**Columnar Reports, 678-681**  
**Common Functions, 183-187**  
**Common Values, 182-183**  
**ComponentOne Copyright Notice, 30**  
**Concepts, 73-74**  
**Conditionally Show or Hide Details, 388-389**  
**Configure HTTPHandlers in IIS 6.x, 427-429**  
**Configure HTTPHandlers in IIS 7.x, 429-432**  
**Connect to a Data Source, 301-302**  
**constant expressions, 336**  
**Container, 140-142**  
**Converting Crystal Reports/MS Access Reports, 42-45**  
**Copy, 740-750**  
**CPL Page Report, 102-104**  
**CPL Report Loader, 451-455**  
**CPL Samples, 444-445**  
**Create a Bullet Graph, 327-328**  
**Create a Drill-Down Report, 348-349**

**Create a Summary Report, 381-383**  
**Create a Whisker Sparkline, 328-329**  
**Create an ALL Parameter, 330-331**  
**Create and Add Themes, 334-335**  
**Create and Edit a Shared Data Source, 304**  
**Create and Use a Master Report, 339-340**  
**Create Common Page Reports, 325**  
**Create Common Section Reports, 379-380**  
**Create Green Bar Report, 327**  
**Create Green Bar Reports, 383-384**  
**Create Red Negatives Report, 326-327**  
**Create Report, 445-446**  
**Create Top N Report, 325-326**  
**Create Top N Reports, 380-381**  
**Creating a Basic End User Report Designer (Pro Edition), 714-718**  
**Cross Section Controls Sample, 467-469**  
**Cross Section Controls, 251-253**  
**Cross Tab Report Sample, 469-470**  
**Cultures, 293-299**  
**Custom Annotation Sample, 479-480**  
**Custom Data Provider, 591-626**  
**Custom HTML Outputer, 651-657**  
**Custom Preview Sample, 480-484**  
**Custom Resource Locator, 221-225 , 440-441**  
**Custom Web Exporting (Std Edition), 646-651**  
**Custom Web Exporting in a Page Report, 586-591**  
**CustomDataProvider, 492-494**  
**Customize and Apply a Theme, 335-336**  
**Customize the FlashViewer Toolbar, 423-426**  
**Customize the Viewer Control, 416-418**  
**Customize, Localize, and Deploy, 413-414**  
**Customizing the Flash Viewer UI, 718-724**  
**Customizing the HTML Viewer UI, 724-727**  
**Data Bar, 212-215**  
**Data Field Expressions Sample, 489-490**  
**Data Sources and Datasets, 173**  
**Data Visualizers, 202**  
**Data, 441 , 456-457**  
**DataSet DataSource, 446-447**  
**DataSet Dialog, 177-179**  
**DataSet Web Service, 733-734**  
**DataSet Windows Application, 727-729**  
**Date, Time, and Number Formatting, 264-265**

**Deploy Web Applications, 421-422**  
**Deploy Windows Applications, 420-421**  
**Deploy, 421-422**  
**Design View, 76-78**  
**Designer Buttons, 81-85**  
**Designer Control (Pro Edition), 290-291**  
**Designer Tabs, 79-81**  
**Digital Signature Sample, 494-495**  
**Display Page Numbers and Report Dates, 373-374**  
**Document Map, 285-286**  
**Document Web Service, 729-731**  
**Document Windows Application, 731-733**  
**Drilldown Reports, 575-576**  
**Drill-Down Reports, 284**  
**Drill-Through Reports, 576-583**  
**Embed Subreports , 402-403**  
**End User Designer Sample, 495-497**  
**End User License Agreement, 30-31**  
**Excel Export, 276-277**  
**Exploring Page Reports , 92-95**  
**Exploring Section Reports , 91-92**  
**Export a Page Report (Export Filter), 340-342**  
**Export a Page Report (Rendering Extension), 342-343**  
**Export a Section Report, 408-410**  
**Export Filters, 270**  
**Exporting, 269-270**  
**Expressions in Reports, 550-552**  
**Expressions, 180-182**  
**Filtering, 283-284**  
**Financial Chart, 641-644**  
**FixedPage Dialog, 191-194**  
**Font Linking, 278-279**  
**FormattedText, 142-144**  
**FPL Page Report, 104-105**  
**FPL Report Loader Sample, 442-444**  
**FPL Samples, 440**  
**Getting Started with the Web Viewer, 55-57**  
**Getting Started, 45**  
**Green Bar, 383-384**  
**Group On Unbound Fields, 686-693**  
**Grouping Data (Page Layout), 194-196**  
**Grouping Data in Section Reports, 261-264**  
**Grouping in a Data Region, 314-321**

**Grouping in a FixedPage, 313-314**  
**How To, 299**  
**HTML Export, 270-272**  
**Hyperlinks and DrillThrough Sample, 484-486**  
**Icon Set, 202-206**  
**IList Binding Sample, 458-460**  
**Image, 144-146**  
**Inherit a Report Template, 384-385**  
**Inheritance Sample, 470-471**  
**Insert or Add Pages, 399-402**  
**Install ActiveReports Developer, 24-25**  
**Installation, 23-24**  
**Installed Files, 25-28**  
**Installing Help in Visual Studio 2010, 28-29**  
**Interactive Features, 279**  
**Interactive Reports, 570**  
**Label, 226-229**  
**Layout Files with Embedded Script, 661**  
**Layout Loader Sample, 471-475**  
**Layout, 442 , 463**  
**Layouts, 451**  
**License Your ActiveReports, 32-38**  
**License, 32-38**  
**Line (Section Report) , 238-239**  
**Line Spacing and Character Spacing, 268-269**  
**Line, 146-147**  
**Linking in Reports, 284-285**  
**LINQ Sample, 460**  
**List, 147-150**  
**Load a File into a RichTextBox Control, 374-378**  
**Localization, 293**  
**Localize ActiveReports Resources, 415-416**  
**Localize Reports, TextBoxes, and Chart Controls , 414-415**  
**Localize the End User Report Designer, 422-423**  
**Localize the Flash Viewer, 426-427**  
**Localize the Viewer Control, 418-420**  
**Localize, 293 , 293-299**  
**Mail Merge with RichText, 693-700**  
**Master Detail Reports, 547-550**  
**Master Reports, 200-202**  
**Matrix Reports, 539-543**  
**Matrix, 150-155**  
**Medium Trust Support, 63-64**

**Memory, 267 , 740-750**  
**Modify Data Sources at Run Time, 369-371**  
**Multiline in Report Controls, 268**  
**Normalized DataSet, 447-448**  
**OleDb DataSource, 448-449**  
**OleObject, 247-248**  
**Optimizing Section Reports, 265-267**  
**Overflow Data in a Single Page, 517-520**  
**Overflow Data in Multiple Pages, 512-517**  
**OverflowPlaceHolder, 155-157**  
**Overlaying Reports (Letterhead), 681-686**  
**Page Report Concepts , 106-107**  
**Page Report How To, 299-301**  
**Page Report Toolbox, 107-108**  
**Page Report Walkthroughs , 507-508**  
**Page Report, 440**  
**Page Reports On Web, 455-456**  
**Page Tabs, 85-86**  
**Page Unbound Data Sample, 441-442**  
**PageBreak, 239-240**  
**Parameterized Reports, 583-586**  
**Parameters, 279-283 , 740-750**  
**PDF Export, 272-274**  
**Picture, 237-238**  
**Preview, 479**  
**Print Methods In ActiveReports Developer, 437-439**  
**Print Multiple Copies, Duplex and Landscape, 386-388**  
**Print Mutliple Pages per Sheet Sample, 486-487**  
**Printing, 740-750**  
**Professional Web Sample, 497-503**  
**Professional, 492**  
**Properties Window, 95**  
**Provide One-Touch Printing in the WebViewer (Pro Edition), 435**  
**Provide PDF Printing in the Silverlight Viewer (Pro Edition), 435-437**  
**Range Bar Progress, 209-212**  
**Range Bar, 206-209**  
**RDF Viewer Sample, 487-488**  
**Recursive Hierarchy Reports, 552-556**  
**Redistributable Files, 31-32**  
**Rendering, 198-200**  
**Report Data Source Dialog, 173-177**  
**Report Dialog , 189-191**  
**Report Explorer, 90-91**

**Report Menu, 78-79**  
**Report Settings Dialog, 260-261**  
**Report Types, 99-102**  
**Report Wizard, 449-450**  
**ReportInfo, 249-251**  
**Reports with Bookmarks, 570-575**  
**Reports with Custom Code, 560-564**  
**Reports with Parameterized Queries, 556-560**  
**Reports with Stored Procedures, 564-566**  
**Reports with XML Data, 543-547**  
**Requirements, 24**  
**RichTextBox, 234-236**  
**RTF Export, 275-276**  
**Rulers , 95-96**  
**Run Time Data Sources, 710-714**  
**Run Time Layouts, 700-710**  
**Run Time or Ad Hoc Reporting, 700**  
**Samples and Walkthroughs, 439**  
**Samples, 439-440**  
**Save and Load RDF Report Files, 410-411**  
**Save and Load RPX Report Files , 411-413**  
**Script for Simple Reports, 661-668**  
**Script for Subreports, 668-676**  
**Scripting in Section Reports, 259-260**  
**Scroll Bars, 96-97**  
**Section Report Concepts , 225**  
**Section Report Events, 255-259**  
**Section Report How To, 362**  
**Section Report Structure, 253-255**  
**Section Report Toolbox, 225-226**  
**Section Report Walkthroughs, 626**  
**Section Report, 456**  
**Set a Drill-Through Link, 349-351**  
**Set a Hidden Parameter, 333-334**  
**Set Detail Grouping In Sparklines, 321-322**  
**Set Filters in Page Reports, 322-325**  
**Set Up Collation, 336-337**  
**Shape (Section Report), 236-237**  
**Shape, 157-158**  
**Shared Data Source (RDSX), 179-180**  
**Shrink Text to Fit in a Control, 291**  
**Side-by-Side Installation, 29-30**  
**Silverlight Viewer Sample, 503-504**

**Single Layout Reports, 508-511**  
**Snap Lines, 97-98**  
**Sort Data, 343-346**  
**Sorting, 286-288**  
**Sparkline, 158-162**  
**Standalone Designer and Viewer, 291-293**  
**Standard Edition Web Sample, 490-492**  
**Style Sheets Sample, 475-476**  
**SubReport (Section Report), 246-247**  
**Subreport in a CPL Report, 523-531**  
**Subreport Sample, 476-479**  
**Subreport Walkthroughs, 628-629**  
**Subreport, 162-164**  
**Subreports with Run-Time Data Sources, 629-633**  
**Subreports with XML Data, 633-637**  
**Summary, 488**  
**Table, 164-169**  
**Text Export, 274-275**  
**Text Justification, 267-268**  
**TextBox (Section Report) , 229-232**  
**TextBox, 169-173**  
**Themes, 197-198**  
**TIFF Export, 277-278**  
**Toolbar, 86-90**  
**Toolbox, 95**  
**Troubleshooting, 740-750**  
**Unbound Chart, 644-646**  
**Unbound Data Sample, 460-462**  
**Upgrading Reports, 38-39**  
**Use Advanced Printing Options, 434-435**  
**Use Constant Expressions in a Theme, 336**  
**Use Custom Controls on Reports, 378-379**  
**Use External Style Sheets, 398-399**  
**Use Fields in Reports, 432-434**  
**Using Script in a Page Report, 187-189**  
**Using the Flash Viewer, 57-60**  
**Using the HTML Viewer , 60-62**  
**Using the Silverlight Viewer, 64-67**  
**Using the Viewer, 48-55**  
**Using the WPF Viewer, 67-72**  
**Viewing Reports, 47-48**  
**Walkthroughs, 506-507**  
**Web Services, 727**

**Web, 421-422 , 490 , 455**

**WebViewer, 435 , 740-750**

**Welcome to ActiveReports Developer 7**

ActiveReports Developer Guide, 13

ActiveReports Editions, 17-23

Breaking Changes, 39-42

ComponentOne Copyright Notice, 30

Converting Crystal Reports/MS Access Reports, 42-45

End User License Agreement, 30-31

Install ActiveReports Developer, 24-25

Installation, 23-24

Installed Files, 25-28

Installing Help in Visual Studio 2010, 28-29

Redistributable Files, 31-32

Requirements, 24

Side-by-Side Installation, 29-30

Upgrading Reports, 38-39

Welcome to ActiveReports Developer 7, 13-14

What's New, 14-17

**Windows Forms Viewer Customization, 289-290**

**Work with Data in Section Reports, 362-363**

**Work with Data, 301**

**Work with Report Controls and Data Regions, 313**

**Work with Report Controls, 371**

**Working with HTML Viewer using Javascript , 62-63**

**WPF Viewer, 737-740**

**Xml Data Provider, 450-451**

**XML Sample, 462-463**

**XML-Based Section Report , 106**

**Zoom Support, 98-99**