

Crystal Reports™ 9

Technical Reference Guide

Crystal Decisions, Inc.
895 Emerson St.
Palo Alto
California, USA 94301

Copyright © 2002 Crystal Decisions, Inc., 895 Emerson St., Palo Alto, California, USA 94301. All rights reserved.

Issue 2.

No part of this documentation may be stored in a retrieval system, transmitted or reproduced in any way, except in accordance with the terms of the applicable software license agreement. This documentation contains proprietary information of Crystal Decisions, Inc., and/or its suppliers.

Trademark Acknowledgements

Crystal Decisions, Crystal Reports, Crystal Enterprise, Crystal Analysis, Crystal Services, Crystal Care, Crystal Assist, Crystal Applications, Info and Holos are trademarks or registered trademarks of Crystal Decisions, Inc. in the U.S. and/or other countries. All other trademarks or registered trademarks referenced are the property of their respective owners.

Contents

Chapter 1: Welcome to Crystal Enterprise Development

About Crystal Enterprise	2
About the guide	2
Product registration	5
Crystal Care technical support	5
Crystal Training	5
Crystal Consulting	6
Document conventions	6

Chapter 2: What's New in the RDC

Report Designer Component 9 (RDC)	8
Connection information property bags	8
Field elements in text objects	8
Cascading Style Sheets	9
Exporting options	9
Printing options	10
Charting options	11
Condition formulas	12
Group name condition formulas	12
Source for linked OLE objects	12
Additional information	12

Chapter 3: Report Designer Component Object Model

Overview of the Report Designer Object Model	14
Unification of the RDC object model	15
Runtime licensing	15
Object naming conflicts	16
Objects and Collections	16
Creatable Objects	16
Application Object	17
Area Object	22
Areas Collection	25
BlobFieldObject Object	26
BoxObject Object	28
ConnectionProperty Object	29
ConnectionProperties Collection	35
CrossTabGroup Object	37
CrossTabGroups Collection	38
CrossTabObject Object	40
Database Object	42
DatabaseFieldDefinition Object	48
DatabaseFieldDefinitions Collection	49
DatabaseTable Object	50
DatabaseTables Collection	53
ExportOptions Object	56
FieldDefinitions Collection	61
FieldElement Object	62

FieldElements Collection	66
FieldMappingData Object	67
FieldObject Object	68
FormattingInfo Object	73
FormulaFieldDefinition Object	74
FormulaFieldDefinitions Collection	75
GraphObject Object	77
GroupNameFieldDefinition Object	83
GroupNameFieldDefinitions Collection	84
LineObject Object	84
MapObject Object	86
ObjectSummaryFieldDefinitions Collection	87
OlapGridObject Object	88
OleObject Object	90
Page Object	92
PageEngine Object	94
PageGenerator Object	97
Pages Collection	103
ParameterFieldDefinition Object	104
ParameterFieldDefinitions Collection	111
ParameterValue Object	112
ParameterValues Collection	112
ParameterValueInfo Object	113
ParameterValueInfos Collection	114
PrintingStatus Object	115

Report Object	116
ReportAlert Object	129
ReportAlerts Collection	130
ReportAlertInstance Object	132
ReportAlertInstances Collection	132
ReportObjects Collection	133
RunningTotalFieldDefinition Object	133
RunningTotalFieldDefinitions Collection	137
Section Object	139
Sections Collection	150
SortField Object	151
SortFields Collection	152
SpecialVarFieldDefinition Object	153
SQLExpressionFieldDefinition Object	154
SQLExpressionFieldDefinitions Collection	155
SubreportLink Object	157
SubreportLinks Collection	158
SubreportObject Object	159
SummaryFieldDefinition Object	162
SummaryFieldDefinitions Collection	164
TableLink Object	165
TableLinks Collection	166
TextObject Object	167
Examples	171
Enumerated Types	201

Chapter 4: Programming the Crystal Report Viewers

Enhancements to the Report Viewer	232
Application Development with Crystal Report Viewers	232
Crystal Report Viewer for ActiveX	233
The Crystal Report Viewer Java Bean	240

Chapter 5: Report Viewer Object Model

Report Viewer/ActiveX Object Model technical reference	244
Enumerated Types	265
The Report Viewer/Java Bean technical reference	267

Chapter 6: Programming the Embeddable Crystal Reports Designer Control

Overview	274
Creating a Microsoft Visual Basic sample application with the Embeddable Designer	274
Creating a Microsoft Visual C++ sample application with the Embeddable Designer	280
Designing reports in the Embeddable Designer	288

Chapter 7: Embeddable Crystal Reports Designer Control Object Model

Overview of the Embeddable Designer	292
Application development with the Embeddable Designer	292
Distributing the Embeddable Designer	293
Properties and methods of the Embeddable Designer	294

Chapter 8: Active Data

Active Data Drivers	298
Crystal Data Object	308
Crystal Data Source Type Library	311
COM data provider	320

Chapter 9: Crystal Data Source Object Models

Crystal Data Source Object Models	328
Crystal Data Objects	328
CrystalComObject	328
Crystal Data Source Type Library	335

Chapter 10: Creating User-Defined Functions in C

Overview of User-Defined Functions in C	340
Programming User-Defined Functions in C	340
Programming the UFL	343
Setting up a UFL project	345
Function definition	346
Returning user-defined errors	351
Obtaining parameter values from the parameter block	352
Picture function—a sample UFL function	353
Module Definition (.def) file	355
UFJOB modules	356
Implement InitJob and TermJob	357
Special functions	358

Chapter 11: Creating User-Defined Functions in Visual Basic	
Overview of User-Defined Functions in Visual Basic	360
Programming User-Defined Functions in Visual Basic	360
Visual Basic and Crystal Reports	365
Sample UFL Automation Server	369
Chapter 12: Creating User-Defined Functions in Delphi 3.0	
Overview of User-Defined Functions in Delphi	372
Programming User-Defined Functions in Delphi	372
Delphi and Crystal Reports	376
Chapter 13: Deprecated and Retired API Reference	
Retired Developer APIs	382
Retired P2smon.dll functions	383
Deprecated RDC Interfaces	386
Deprecated RDC Properties	388
Deprecated RDC Methods	390
Deprecated RDC Export Format Types	394
Index	395

Welcome to Crystal Enterprise Development

1

Welcome! This chapter introduces you to developing with Crystal Enterprise, the world standard for desktop and web reporting, and provides you with an overview of the contents of this guide.

About Crystal Enterprise

Crystal Enterprise is designed to work with your database to help you analyze and interpret important information. Crystal Enterprise makes it easy to create simple reports; it also has the comprehensive tools you need to produce complex or specialized reports.

Create any report you can imagine

Crystal Enterprise is designed to produce the report you want from virtually any data source. Built-in report wizards and experts guide you step by step through building reports and completing common reporting tasks. Formulas, cross-tabs, subreports, and conditional formatting help make sense of data and help you uncover important relationships that might otherwise be hidden. Geographic maps and charts communicate information visually when words and numbers are simply not enough.

Extend reporting to the Web

The flexibility of Crystal Enterprise doesn't end with creating reports—your reports can be published in a variety of formats including Microsoft® Word and Excel, email, and even over the Web. Advanced Web reporting lets other members of your workgroup view and update shared reports inside their web browser.

Incorporate reports into applications

Application and web developers can save time and meet the needs of their users by integrating the report processing power of Crystal Enterprise into their database applications. Support for most popular development languages makes it easy to add reporting to any application.

Whether it's the web master in IT, the promotion manager in marketing, the database administrator in finance, or the CEO, Crystal Enterprise is a powerful tool designed to help everyone analyze and interpret the information that's important to them.

About the guide

The guide includes technical information on the development tools included with Crystal Enterprise that allow you to integrate reporting into your application. The guide covers the Report Designer Component. It also contains tutorials and samples that will further aid you with adding the power of Crystal Enterprise to your application.

Chapter contents

The following is a short description of each chapter in this guide.

Chapter 1: Welcome to Crystal Enterprise Development

Welcome! This chapter introduces you to developing with Crystal Enterprise, the world standard for desktop and web reporting, and provides you with an overview of the contents of this guide.

Chapter 2: What's New in the RDC

With the release of Crystal Enterprise version 9, Crystal Enterprise continues to improve the flexibility and power of the Report Designer Component. New features allow you to enhance the reporting experience of your users.

This chapter introduces the new features and enhancements.

Chapter 3: Report Designer Component Object Model

The Report Designer Component is the ultimate development tool for your reporting needs. In this chapter you will find a detailed description of the Report Designer Component Object Model and its properties, methods and events.

Chapter 4: Programming the Crystal Report Viewers

The Crystal Report Viewer is a front-end user interface for viewing reports. In this chapter you will find detailed information on implementing the ActiveX and Java Bean viewers in your application.

Chapter 5: Report Viewer Object Model

The Crystal Report Viewer contains an extensive object model allowing you complete control over how the viewer appears and functions. This chapter provides detailed information on the properties, methods and events of the Crystal Report Viewer object model for both the ActiveX viewer, and the Java Bean viewer.

Chapter 6: Programming the Embeddable Crystal Reports Designer Control

This chapter demonstrates how to integrate the Embeddable Crystal Reports Designer Control (Embeddable Designer) into your application. Included are tutorials on creating sample applications in both Microsoft Visual Basic and Microsoft Visual C++. A brief section on creating reports in the designer at runtime is also included.

Chapter 7: Embeddable Crystal Reports Designer Control Object Model

The Embeddable Crystal Reports Designer Control (Embeddable Designer) is a new addition to the Report Designer Component; it enables you to provide a Crystal Reports designer in your application. In this chapter, you will find a general overview of the control as well as detailed information on its properties and methods.

Chapter 8: Active Data

This chapter provides information on using active data in the Crystal Reports Development environment. Four areas are covered: the active data drivers, the Crystal Data Object, the Crystal Data Source Type Library, and the COM Data Provider. By reading these sections you will learn how to create active data reports, create recordsets, and connect to the data source through the Report Designer Component automation server.

Chapter 9: Crystal Data Source Object Models

Crystal Reports provides support for reporting off data when no true data source exists. In this chapter you will find detailed information, including properties and methods, on Crystal Data Objects and the Crystal Data Source Type Library.

Chapter 10: Creating User-Defined Functions in C

Crystal Reports allows you to create User-Defined Functions that are recognized by the Crystal Reports Formula Editor. In this chapter you will find detailed information on programming User-Defined Functions in C.

Chapter 11: Creating User-Defined Functions in Visual Basic

Crystal Reports allows you to create User-Defined Functions that are recognized by the Crystal Reports Formula Editor. In this chapter you will find detailed information on programming User-Defined Functions in Microsoft Visual Basic.

Chapter 12: Creating User-Defined Functions in Delphi 3.0

Crystal Reports allows you to create User-Defined Functions that are recognized by the Crystal Reports Formula Editor. In this chapter you will find detailed information on programming User-Defined Functions in Delphi.

Chapter 13: Deprecated and Retired API Reference

The following developer APIs, interfaces, methods, and properties are retired or deprecated. Retired features are no longer available or supported. Deprecated features, while they may be supported for backwards compatibility, are not recommended for use in the current version of Crystal Reports. Deprecated features may become obsolete in future versions of the product.

Product registration

There are several ways you can register your product:

- Fill out the Product Registration form on the Crystal Decisions, Inc. web site at:
<http://www.crystaldecisions.com/register/>
- Print the Product Registration form and fax it to the registration fax number closest to you. Crystal Decisions will then fax you a registration number that can be entered into the product the next time you use it.

Registration fax numbers

USA/Canada +1 (604) 681-5147

United Kingdom +44 (0) 20 8231 0601

Australia +6 2 9955 7682

Germany +49 (0) 69 9509 6182

Hong Kong +852 2893 2727

Singapore +65 777 8786

Registering the product ensures that you are kept up-to-date with product advancements.

Crystal Care technical support

To find out about the technical support programs available for Crystal Enterprise:

- Consult the enclosed Crystal Care information card.
- Go to our support web site at:
<http://support.crystaldecisions.com/crystalcare/>
- Contact your regional office. For details, go to:
<http://www.crystaldecisions.com/contact/offices.asp>

Crystal Training

Whether you're a developer, information technology professional, or business user, we offer a wide range of Crystal Enterprise training courses designed to build or enhance your existing skills. Courses are available online, at certified training centers, or at your own site:

- For a complete list of training courses and special offers, visit:
<http://www.crystaldecisions.com/training/>
- Or contact your regional office. For details, go to:
<http://www.crystaldecisions.com/offices/>

Crystal Consulting

Our global team of certified consultants and consulting partners can guide you through a corporate-wide solution—including strategy, design, integration and deployment—for the fastest results, maximum performance, and increased productivity.

- To learn more, visit:
<http://www.crystaldecisions.com/consulting/>
- Or contact your regional office. For details, go to:
<http://www.crystaldecisions.com/offices/>

Document conventions

This guide uses the following conventions:

- Commands and buttons
For easy recognition within procedures, User Interface (UI) features appear in bold type. For example: On the **File** menu, click **New**.
- Keyboard shortcuts
Delete means the Delete key, or the Del key on your numeric keypad. Enter means the Enter, Return, or CR key, depending on which of these keys appears on your keyboard.
- Key combinations
CTRL+KEY, SHIFT+KEY, and ALT+KEY are examples of key combinations. Hold down the first key in the combination and, at the same time, press the second key in the combination (designated above as KEY). For example: CTRL+C means hold the Control key down and press the letter C on your keyboard (CTRL+C is the Windows Copy command).
- Monospaced font indicates data that you enter using your keyboard. For example: In the Formula Editor, type `If Sales > 1000 Then crRed`. It is also used to show sample code: `CRViewer1.ReportSource = CRXReport`

With the release of version 9, Crystal Enterprise continues to improve the flexibility and power of the Report Designer Component. New features allow you to enhance the reporting experience of your users.

This chapter introduces the new features and enhancements.

Report Designer Component 9 (RDC)

Designed for Microsoft Visual Studio and other COM-based development environments, the feature-rich Report Designer Component 9 (RDC) gives developers unprecedented control over report layout and formatting for web and Windows application reporting.

Reflecting the enhanced features of the Crystal Report Engine, the RDC supports most of the new Crystal Enterprise features, including database connectivity through connection information property bags, fields in text objects, cascading style sheets, additional export options, additional printing options, additional charting options, conditional formulas for areas sections and report objects, group name condition formulas, and retrieving the path to the data source for OLE objects.

Connection information property bags

The new query engine for Crystal Reports introduces a new feature: each database DLL has its own unique set of custom properties. These properties allow the developer to create a dynamic user interface, based on the data sources used by the report.

The **DatabaseTable Object** returns the **ConnectionProperties Collection** through the new **ConnectionProperties** property. Each **ConnectionProperty Object** represents one of the custom property bags of the database DLL. The property bag contains such things as the name and value of the property. Using these properties, you can read the connection information for the report and then write back new connection information to your report.

Note: Many methods and properties of the **DatabaseTable Object**, used for connecting to a report's data source, have been replaced by the properties available in the **ConnectionProperty Object**. For a complete list of these deprecated calls, see “**Deprecated RDC Properties**” on page 388 and “**Deprecated RDC Methods**” on page 390.

Field elements in text objects

You can now exercise greater control over text objects by adding and removing fields at runtime. You are also able to exercise the same control over the field in the text object as you can with a field outside the text object.

The **TextObject Object** returns the **FieldElements Collection** through the new **FieldElements** property. Using this collection, you can add or delete an embedded field. Each **FieldElement Object** in the collection provides properties and methods that you can use to customize the embedded field within the text object.

Cascading Style Sheets

At runtime, you can add a cascading style sheet (CSS) class to each report object and section ([Section Object](#)) in a report. Report objects include [BlobFieldObject Object](#), [FieldObject Object](#), [GraphObject Object](#), and so on. The (CSS) class is used to apply formatting to reports viewed over the Web.

Generally, when you create a report, you format a particular report object (for example, a database field or a text object) within the Crystal Report Designer. When the report is rendered into HTML and viewed over the web, these formatting options are automatically converted to HTML. To change the formatting of the HTML page, you would need to make a change to the report itself.

The CSS class is set through the `CSSClass` property, found in the `Section` object and each of the report objects. This enables you to apply classes to report objects and to sections of a report, instead of manually changing the formatting of the report. You specify the name of the class, and its scope, in the Crystal Report Designer. Then you specify the value of the class through an external style sheet. The external style sheet is then applied to the report when the report is rendered to HTML.

Exporting options

There have been many enhancements to the exporting capability of Crystal Enterprise in this version. The RDC lets you program the new exporting features through properties that have been added to the [ExportOptions Object](#).

Paginated export to Excel, PDF, RTF, and Word formats

Export your reports—in whole or in part—directly to the popular Portable Document Format (PDF), and then distribute these reports by email, over the Web, or in print. Alternatively, take advantage of the new Rich Text Format (RTF) exporting feature, which is based on the Crystal Reports Encapsulated Page Format (EPF).

In addition, Crystal Reports now supports page-ranged export to Excel, PDF, RTF, and Word formats.

Note: Paginated export to RTF and PDF was introduced in Crystal Reports 8.5.

New properties have been added to the [ExportOptions Object](#) to support these new features, they are:

- `ExcelExportAllPages`
- `ExcelFirstPageNumber`
- `ExcelLastPageNumber`
- `PDFExportAllPages`
- `PDFFirstPageNumber`
- `PDFLastPageNumber`
- `RTFExportAllPages`

- RTFFirstPageNumber
- RTFLastPageNumber.
- WORDWExportAllPages
- WORDWFirstPageNumber
- WORDWLastPageNumber

Additional export options

Additional properties have been added to the **ExportOptions Object** to enhance exporting to Excel, text formats, and mail destinations.

ExcelConvertDateToString property

Use this property to convert data and date-time fields to string fields when exporting to Excel.

ExcelPageBreaks property

Use this property to add page breaks when exporting to Excel. The page break is in the form of a blank row of cells between exported pages.

MailPassword property and MailUserName property

Use these properties when exporting to the MAPI destination when you are required to provide a user name and password.

UseDefaultCharactersPerInch property and UserDefinedCharactersPerInch property

Use these properties to set the characters per inch when exporting to text. You can allow the default characters per inch by setting the UseDefaultCharactersPerInch property to True, or set your own value through the UserDefinedCharactersPerInch property.

Printing options

Crystal Enterprise continues to improve printing capabilities by adding new methods to the **Report Object**.

Print to a file

You can now export your report to a printer file using the **PrintOutEx Method (Report Object)**.

Set a custom paper size

You can now set a custom paper size for your report using the **SetUserPaperSize Method (Report Object)**.

Charting options

The following additional properties have been added to the **GraphObject Object** to enhance charting in the RDC. Refer to the *Crystal Reports User's Guide* for more information on charting in Crystal Enterprise.

Automatically scale series axis

The runtime property, `EnableAutoScaleSeriesAxis`, has been added so that you can automatically scale a series axis.

Set the chart axis title

New runtime properties have been added to allow you to define a title for a chart axis. The new properties are: `XAxisTitle`, `YAxisTitle`, and `ZAxisTitle`.

Default titles for charts

Previously, when you used the Chart Expert to create a chart, the expert didn't assign a title to the chart unless you specified one. Now, the expert supplies the following titles by default: title, subtitle, footnote, group title, data title, data2 title, series title, x axis title, y axis title, and z axis title.

To support these enhancements, the following new properties have been added to the **GraphObject Object** for the Report Designer Component at runtime:

- `IsTitleByDefault*`
- `IsSubTitleByDefault*`
- `IsFootnoteByDefault*`
- `IsGroupTitleByDefault*`
- `IsSeriesTitleByDefault*`
- `IsDataTitleByDefault*`
- `IsData2TitleByDefault*`
- `IsXAxisTitleByDefault`
- `IsYAxisTitleByDefault`
- `IsZAxisTitleByDefault`

* Added in Crystal Enterprise version 8.5.

Automatically set the series axis range

The new runtime property, `AutoRangeSeriesAxis`, allows you to automatically set the series axis range.

Set the minimum and maximum series axis values

You can now set the minimum and maximum series axis values using the new runtime properties: `MaxSeriesAxisValue` and `MinSeriesAxisValue`.

Set the series axis division method, division number, and number format

You can now set the series axis division method, division number, and number format using the new runtime properties: `SeriesAxisDivisionMethod`, `SeriesAxisDivisionNumber`, and `SeriesAxisNumberFormat`.

Condition formulas

In version 9, you can now set condition formulas for all report objects, areas ([Area Object](#)), and sections ([Section Object](#)) in a report at runtime. Report objects include [BlobFieldObject Object](#), [FieldObject Object](#), [GraphObject Object](#), and so on. This new feature allows you to set the conditional formula for each of the attributes found in the Format Editor dialog box. This offers you full conditional control over the appearance of the report object area and section. For more information, see “Format Editor dialog box” in the *Crystal Reports Online Help* (*craw.chm*).

The conditions for the Area object, Section object, and report objects, are set through the `ConditionFormula` property. The type of condition formula is set as a parameter, and the formula is passed as a string.

Group name condition formulas

You can now customize the group names displayed in your report through the `GroupNameConditionFormula` of the [GroupNameFieldDefinition Object](#).

Source for linked OLE objects

In this version of the RDC, you can now retrieve the path and file name of a linked OLE object using the [GetLinkSource Method](#) ([OleObject Object](#)).

Additional information

Crystal Reports product news

<http://www.crystaldecisions.com/products/crystalreports/>

Crystal Reports demos

<http://www.crystaldecisions.com/products/crystalreports/showme/>

Crystal Enterprise product news

<http://www.crystaldecisions.com/products/crystalenterprise/>

Product information

<http://www.crystaldecisions.com/products/>

Developer Zone

http://www.crystaldecisions.com/products/dev_zone/

Online support, samples and tech briefs

<http://support.crystaldecisions.com/homepage/>

Training and consulting

<http://www.crystaldecisions.com/services/>

Crystal Decisions homepage

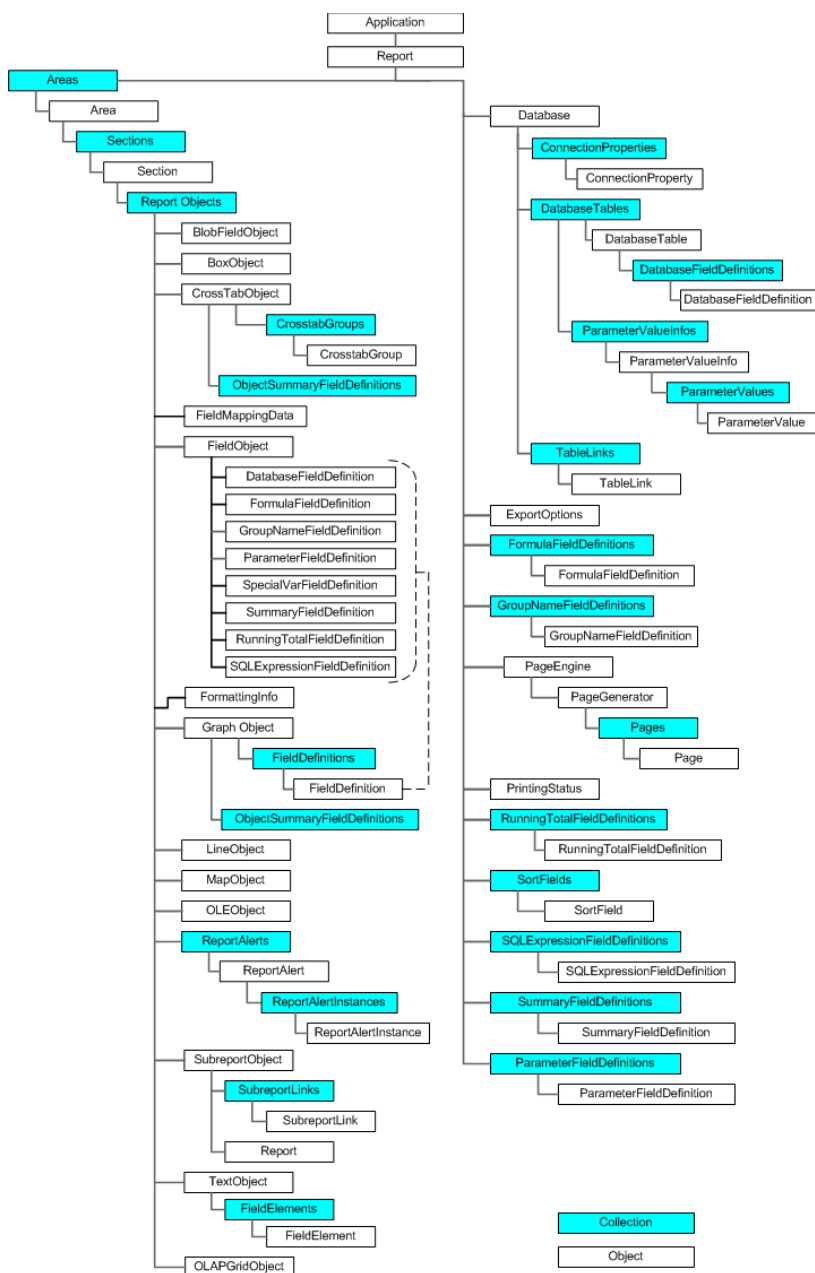
<http://www.crystaldecisions.com/homepage/>

Report Designer Component Object Model

3

The Report Designer Component is the ultimate development tool for your reporting needs. In this chapter you will find a detailed description of the Report Designer Component Object Model and its properties, methods, and events.

Overview of the Report Designer Object Model



Unification of the RDC object model

Craxddrt.dll (Crystal Reports ActiveX Designer Design and Runtime Library) is a new unified object model that combines the runtime capabilities of the Craxdrt.dll (Crystal Reports ActiveX Designer Run Time Library) with the design time capabilities of the Craxddt.dll (Crystal Reports ActiveX Designer Design Time Library). Craxddrt.dll will replace Craxddt.dll for versions 8.5 and up. Both the Craxddrt.dll and the Craxdrt.dll contain all the objects and associated methods, properties, and events needed for creating, opening, exporting, saving, and printing a report at run time. In addition, Craxddrt.dll is either used with the RDC ActiveX Designer when designing reports at design time, or used with the Embeddable Designer when designing reports at run time. See “[Embeddable Crystal Reports Designer Control Object Model](#)” on page 291 for more information.

Note: The RDC ActiveX Designer is only available in Microsoft Visual Basic.

Prior to version 8.5, the Craxdrt.dll would be distributed with an application. Now the developer has a choice of two automation servers to distribute. Craxdrt.dll is backwards-compatible with previous versions and contains all the new features introduced in this version. Use the Craxdrt.dll for any client-side application that does not contain the Embeddable Designer, or use it for any server-side application. Craxddrt.dll is apartment-model threaded, but is not thread safe, and can only be used in a client-side application. Although the Craxddrt.dll is a fully functional automation server for the RDC, and can work in any client-side application, it will increase the install size. Therefore, it is recommended that you only use Craxddrt.dll with the Embeddable Crystal Reports Designer Control.

Runtime licensing

Your Crystal Reports developer edition license provides the rights to distribute software applications that use or embed *most* files and functionality of the product, free of charge. In addition, your Crystal Reports developer edition contains new features that provide a great deal more flexibility and power for developers to embed reporting into their applications. These runtime report creation features may only be used and distributed if the appropriate licensing is obtained from Crystal Decisions by paying the required royalty fees. Please refer to the *License Manager Help* (*License.hlp*) for lists of features that are royalty-free and royalty-required.

Object naming conflicts

If your project includes other libraries that contain objects named identical to those found in the Crystal Report Engine Object Library, you will encounter conflicts unless you reference the objects using a prefix. For example, if you have included the DAO Library with the Crystal Report Engine Object Library in your project, both libraries include a Database Object. In order to avoid conflicts, you must prefix the objects as follows:

`CRAXDRT.Database`

for the Crystal Report Engine Object Library, or

`DAO.Database`

for the DAO Library.

Objects and Collections

The following Objects and Collections, listed alphabetically, are discussed in this section. Properties, Methods, and Events are listed under the appropriate Object or Collection.

Note: Some of the new report creation calls are not included in the free runtime license included with Crystal Reports. For a list of calls that require additional licensing, please see the *License Manager Help (License.hlp)*.

Creatable Objects

There are several Crystal Reports objects which can be created by passing their Prog Ids into the CreateObject function provided by Visual Basic. The table below lists these objects.

Object	Prog Id
Application Object	CrystalRuntime.Application
ParameterValue Object	CrystalRuntime.ParameterValue
ParameterValues Collection	CrystalRuntime.ParameterValues
ParameterValueInfo Object	CrystalRuntime.ParameterValueInfo
ParameterValueInfos Collection	CrystalRuntime.ParameterValueInfos

Note: The version number can be appended to the Prog Id (for example, `CrystalRuntime.ParameterValue.9`). If the version number is not specified, CreateObject will create the object using the newest version of `craxdrt.dll`.

Application Object

An instance of the Application object can be created using the Visual Basic New keyword or the CreateObject function and the Prog Id CrystalRuntime.Application.9. For example:

- Using the New keyword

```
Dim app as New CRAXDRT.Application
Set app = New CRAXDRT.Application
```

Or,

```
' Automatically creates a new instance of the object when it is first
' referenced in the code, so it doesn't have to be set.
Dim app as New Application
```

- Using the CreateObject function

```
' If the version number of the application is not specified, CreateObject
' will create an application running against the new version of
' craxdrt.dll
Dim app As Application
Set app = CreateObject("CrystalRuntime.Application")
```

Or,

```
' Specify the version number to require version 8.5 of the dll.
Dim app As Application
Set app = CreateObject("CrystalRuntime.Application.8.5")
```

Application Object Methods

The following methods are discussed in this section:

- “CanClose Method (Application Object)” on page 17
- “GetLicenseStatus Method (Application Object)” on page 18
- “GetVersion Method (Application Object)” on page 18
- “LogOffServer Method (Application Object)” on page 18
- “LogOnServer Method (Application Object)” on page 19
- “LogOnServerEx Method (Application Object)” on page 20
- “NewReport Method (Application Object)” on page 21
- “OpenReport Method (Application Object)” on page 21
- “SetMatchLogOnInfo Method (Application Object)” on page 22

CanClose Method (Application Object)

The CanClose method indicates whether or not the “Application Object” can be destroyed. This method will return FALSE as long as there are valid Report objects in existence and at least one Report Object is in the printing-in-progress state. The Application object can only be destroyed if no instances of the “Report Object” are in the printing-in-progress state. If you obtain a Report object directly from the Report Designer Component added to your project at design time, then CanClose will always return False until you destroy that object (usually by setting it equal to Nothing).

Syntax

```
Function CanClose () As Boolean
```

Returns

- TRUE if the Engine can be closed.
- FALSE if the Engine is busy.

GetLicenseStatus Method (Application Object)

The GetLicenseStatus method is used to get the license number of the Report Creation API License and the current count of licenses in use.

Syntax

```
GetLicenseStatus(pMaxLicenseNumber, pLicenseUsed) As Boolean
```

Returns

- True if the license is active and the pLicenseUsed parameter was used in the call.
- False if the license is not active and the pLicenseUsed parameter was used in the call.

GetVersion Method (Application Object)

The GetVersion method returns an integer that, when converted to hex, represents the version number of the dll.

Syntax

```
Function GetVersion () As Integer
```

Returns

- Returns an integer that represents the version number of the dll when converted to hexadecimal.

Sample

```
'Display the version number in a text box.  
Text1.text = Hex(CRXApplication.GetVersion)
```

LogOffServer Method (Application Object)

The LogOffServer method logs off an SQL server or other data sources such as ODBC or OLEDB provider. Use this method when you have logged on to the data source using “[LogOnServer Method \(Application Object\)](#)”.

Syntax

```
Sub LogOffServer (pDllName As String, pServerName As String,  
    [pDatabaseName], [pUserID], [pPassword])
```

Parameters

Parameter	Description
pDLLName	Specifies the name of the DLL for the server or password protected non-SQL table you want to log on to, for example "PDSODBC.DLL". Note that the DLLName must be enclosed in quotes. DLL names have the following naming convention: PDB*.DLL for standard (non-SQL) databases, PDS*.DLL for SQL/ODBC databases.
pServerName	Specifies the log on name for the server used to create the report. (For ODBC, use the data source name.) This value is case-sensitive. See Remarks below.
[pDatabaseName]	Specifies the name for the database used to create the report. See Remarks below.
[pUserID]	Specifies the User ID number necessary to log on to the server. See Remarks below.
[pPassword]	Specifies the password necessary to log on to the server.

Remarks

- For parameters pServerName, pDatabaseName, and pUserId, pass an empty string ("") to preserve the existing setting or pass a non-empty string (for example, "Server A") to override a value that is already set in the report.
- If you try to log off a server that is still in use (i.e., there is an object variable still in focus that holds reference to a report that requires being logged on to the server to access data) you will be unable to log off. This will apply to every object that comes from the "Report Object", as they all hold reference to the report through their respective Report properties.
- If you assign the Report object to the ReportSource property of the "CRViewer Object (CRVIEWERLib)", in the Crystal Reports Report Viewer, enabling the report to be displayed through the Report Viewer, you cannot call LogOffServer for the report until you assign a new report to the Report Viewer or close the CRViewer object.

LogOnServer Method (Application Object)

The LogOnServer method logs on to an SQL server or other data sources such as ODBC. Once logged on using this method, you will remain logged on until you call "LogOffServer Method (Application Object)", or until the "Application Object", is destroyed and craxdrt.dll is unloaded.

Syntax

```
Sub LogOnServer (pDllName As String, pServerName As String,  
                [pDatabaseName], [pUserID], [pPassword])
```

Parameters

Parameter	Description
pDLLName	Specifies the name of the DLL for the server or password protected non-SQL table you want to log on to, for example "PDSODBC.DLL". Note that the dllName must be enclosed in quotes. DLL names have the following naming convention: PDB*.DLL for standard (non-SQL) databases, PDS*.DLL for SQL/ODBC databases.
pServerName	Specifies the log on name for the server used to create the report. (For ODBC, use the data source name.) This value is case-sensitive. See Remarks below.
[pDatabaseName]	Specifies the name for the database used to create the report. See Remarks below.
[pUserID]	Specifies the User ID number necessary to log on to the server. See Remarks below.
[pPassword]	Specifies the password necessary to log on to the server.

*When you pass an empty string ("") for this parameter, the program uses the value that's already set in the report. If you want to override a value that's already set in the report, use a non-empty string (that is, "Server A").

Remarks

For parameters pServerName, pDatabaseName, and pUserId, pass an empty string ("") to preserve the existing setting or pass a non-empty string (for example, "Server A") to override a value that is already set in the report.

LogOnServerEx Method (Application Object)

The LogOnServer method logs on to an SQL server or other data sources such as ODBC. Using LogOnServerEx, you can pass server type or connection information. Once logged on using this method, you will remain logged on until you call "[LogOffServer Method \(Application Object\)](#)", or until the "[Application Object](#)", is destroyed and craxdr.dll is unloaded.

Syntax

```
Sub LogOnServerEx (pDllName As String, pServerName As String,
    [pDatabaseName], [pUserID], [pPassword],
    [pServerType], [pConnectionString])
```

Parameters

Parameter	Description
pDLLName	Specifies the name of the DLL for the server or password protected non-SQL table you want to log on to, for example "PDSODBC.DLL". Note that the dllName must be enclosed in quotes. DLL names have the following naming convention: PDB*.DLL for standard (non-SQL) databases, PDS*.DLL for SQL/ODBC databases.
pServerName	Specifies the log on name for the server used to create the report. (For ODBC, use the data source name.) This value is case-sensitive. See Remarks below.
[pDatabaseName]	Specifies the name for the database used to create the report. See Remarks below.
[pUserID]	Specifies the User ID number necessary to log on to the server. See Remarks below.
[pPassword]	Specifies the password necessary to log on to the server.
[pServerType]	Specifies the database Server Type.
[pConnectionString]	Specifies the connection string.

NewReport Method (Application Object)

The NewReport method creates a new empty Report Object.

Syntax

```
Function NewReport () As Report
```

Returns

- Returns a new empty Report Object.

OpenReport Method (Application Object)

The OpenReport method opens an existing report file, creating an instance of the Report object. Through the "**Report Object**", you can change formatting, formulas, selection formulas, and sort fields for the report, then print, preview, or export the report.

Syntax

```
Function OpenReport (pFileName As String, [OpenMethod]) As Report
```

Parameters

Parameter	Description
pFileName	Specifies a string value indicating the file name and path of the report that you want to open.
OpenMethod	"CROpenReportMethod". Specifies whether you want to open the report exclusively. If you do not provide this parameter the report is opened exclusively and it cannot be opened a second time.

Returns

- Returns an instance of the "Report Object" if the report was successfully opened.
- Returns 0 if the report file does not exist or if an error occurs.

Remarks

- The OpenMethod parameter can take the values "0" (open by default) or "1" (open temporary copy).

SetMatchLogOnInfo Method (Application Object)

The SetMatchLogOnInfo Method sets global match log on info option, matching the log on password.

Syntax

```
Sub SetMatchLogOnInfo (b1 As Boolean)
```

Parameter

Parameter	Description
b1	Specifies whether the option is selected (TRUE).

Area Object

The Area Object represents an area in a report. An area is a group of like sections in the report (i.e., Details A - Da, Details B - Db, etc.) that all share the same characteristics. Each section within the area can be formatted differently. This object allows you to retrieve information and set options for a specified area in your report.

Area Object Properties

Property	Description	Read/Write	Restriction in event handler
CopiesToPrint	Integer. Gets or sets the number of copies of each item in the Details section of the report. For example, by default, each line of the Details section only prints once. By setting this to 3, each line of the Details section would print 3 times.	Read/Write	Can be written only when formatting idle.
DetailHeight	Long. Gets the mailing label report detail area height, in twips.	Read only	None
DetailWidth	Long. Gets multiple column report detail area width, in twips.	Read only	None
DiscardOtherGroups	Boolean. Gets or sets the discard other groups option.	Read/Write	
EnableHierarchicalGroupSorting	Boolean. Gets or sets group hierarchically flag.	Read/Write	Formatting idle.
GroupCondition	"CRGroupCondition" . Gets or sets the group condition.	Read/Write	Can be written only when formatting idle.
GroupConditionField	Object. Gets or sets the group condition field.	Read/Write	Can be written only when formatting idle.
GroupIndent	Long. Gets or sets group indent, in twips.	Read/Write	Formatting idle.
GroupNumber	Integer. If the area is a group area, this returns the group number. Otherwise, exception is thrown.	Read only	None
HideForDrillDown	Boolean. Gets or sets hide for drill down option.	Read/Write	Can be written only when formatting idle.
HorizontalGap	Long. Gets the horizontal gaps going across page in a multiple column report.	Read only	None
InstanceIDField	A field definition object. Gets the instance ID field. See Remarks .	Read only	Formatting idle.
KeepGroupTogether	Boolean. Gets or sets the keep group together option.	Read/Write	Can be written only when formatting idle.
KeepTogether	Boolean. Gets or sets the keep area together option.	Read/Write	Can be written only when formatting idle.
Kind	"CRAreaKind" . Gets which kind of area (for example, Details, Report Header, Page Footer, etc.).	Read only	None
Name	String. Gets or sets the area name.	Read/Write	Can be written only when formatting idle.

Property	Description	Read/Write	Restriction in event handler
NewPageAfter	Boolean. Gets or sets the new page after options.	Read/Write	Can be written only when formatting idle.
NewPageBefore	Boolean. Gets or sets the new page before option.	Read/Write	Can be written only when formatting idle.
NumberOfTopOrBottomNGroups	Integer. Gets or sets the number of top or bottom groups.	Read/Write	Formatting idle.
Parent	"Report Object" . Gets reference to the parent object.	Read Only	None
ParentIDField	A field definition object. Gets the parent ID field. See Remarks .	Read only	Formatting idle.
PrintAtBottomOfPage	Boolean. Gets or sets the print at bottom of page option.	Read/Write	Can be written only when formatting idle.
RepeatGroupHeader	Boolean. Gets or sets the repeating group header option.	Read/Write	Can be written only when formatting idle.
ResetPageNumberAfter	Boolean. Gets or sets the reset page number after option.	Read/Write	Can be written only when formatting idle.
Sections	"Sections Collection" . Gets a Collection of all the sections in the area.	Read Only	None
SortDirection	"CRSortDirection" . Gets or sets the group sort direction.	Read/Write	Can be written only when formatting idle.
SpecifiedGroups	Gets or sets the specified groups.	Read/Write	Can be written only when formatting idle.
Suppress	Boolean. Gets or sets the area visibility.	Read/Write	Can be written only when formatting idle.
TopOrBottomNGroupSortOrder	"CRTopOrBottomNGroupSortOrder" . Gets or sets the top or bottom N group sort order.	Read/Write	Formatting idle.
TopOrBottomNSortField	"SummaryFieldDefinition Object" . Gets or sets the top or bottom n sort field.	Read/Write	Formatting idle.

Remarks

After getting the field definition object from the InstanceIDField or the ParentIDField property, you can use the Kind property to determine what type of field definition object it is. All field definition objects have the Kind property.

Area Object Methods

The following methods are discussed in this section:

- ["SetInstanceIDField Method \(Area Object\)"](#) on page 25
- ["SetParentIDField \(Area Object\)"](#) on page 25

SetInstanceIDField Method (Area Object)

Use the SetInstanceIDField method to set an instance ID field.

Syntax

```
Sub SetInstanceIDField (InstanceIDField)
```

Parameter

Parameter	Description
InstanceIDField	Specifies the instance ID field that you want to set.

SetParentIDField (Area Object)

Use the SetParentIDField method to set the parent ID field.

Syntax

```
Sub SetParentIDField (ParentIDField)
```

Parameter

Parameter	Description
ParentIDField	Specifies the parent ID field that you want to set.

Areas Collection

The Areas Collection contains the area objects for every area in the report. Access a specific “Area Object” in the collection using the Item property.

Areas Collection Properties

Property	Description	Read/Write	Restriction in event handler
Count	Long. Gets the number of areas in the collection.	Read only	None
Item (index)	“Area Object”. Gets an item from the Collection. Item has an index parameter that can be either a string reference to the area (for example, “RH”, “PH”, “GHn”, “D”, “GFn”, “PF”, or “RF”) or a numeric, 1-based index (for example, Item (1) for the Report Header area). The items in the collection are indexed in the order they are listed for each section/area.	Read only	None
Parent	“Report Object”. Gets reference to the parent object.	Read only	None

Remarks

Instead of using the Item property as shown, you can reference an area directly (for example, Areas("RH") or Areas(1)).

BlobFieldObject Object

The BlobFieldObject Object allows you to get and set information for bitmap database fields in a report.

BlobFieldObject Object Properties

Examples

"How to format a blob field object" on page 182

Property	Description	Read/Write	Restriction in event handler
BackColor	OLE_COLOR. Gets or sets the object background color.	Read/Write	Can be written only when formatting idle or active.
BorderColor	OLE_COLOR. Gets or sets the object border color.	Read/Write	Can be written only when formatting idle or active.
Bottom Cropping	Long. Gets or sets bottom cropping size, in twips.	Read/Write	Can be written only when formatting idle.
BottomLineStyle	"CRLLineStyle". Gets or sets the bottom line style.	Read/Write	Can be written only when formatting idle or active.
CloseAtPage Break	Boolean. Gets or sets the close border on page break.	Read/Write	Can be written only when formatting idle or active.
Condition Formula	String. Gets or sets the condition formula. The condition formula can be based on recurring records or on summary fields, but it cannot be based on print-time fields, such as running totals or print-time formulas. Condition formulas cannot have shared variables.	Read/Write	Can be written only when formatting idle.
CssClass	String. Gets or sets the cascading style sheet Class.	Read/Write	Can be written only when formatting idle.
Field	"DatabaseFieldDefinition Object". Gets the database field definition object containing information about the BLOB field.	Read only	None
HasDrop Shadow	Boolean. Gets or sets the border drop shadow option.	Read/Write	Can be written only when formatting idle or active.

Property	Description	Read/Write	Restriction in event handler
Height	Long. Gets or sets the object height, in twips.	Read/Write	Can be written only when formatting idle or active.
KeepTogether	Boolean. Gets or sets the keep object together option.	Read/Write	Can be written only when formatting idle or active.
Kind	"CROBJECTKind". Gets CROBJECTKind which specifies the kind of object (for example, box, cross-tab, field, etc.).	Read only	None
Left	Long. Gets or sets the object upper left position, in twips.	Read/Write	Can be written only when formatting idle or active.
LeftCropping	Long. Gets or sets the left cropping size, in twips.	Read/Write	Can be written only when formatting idle.
LeftLineStyle	"CRLINEStyle". Gets or sets the left line style.	Read/Write	Can be written only when formatting idle or active.
Name	String. Gets or sets the object name.	Read/Write	Can be written only when formatting idle.
Parent	"Section Object". Gets reference to the parent object.	Read only	None
RightCropping	Long. Gets or sets the right cropping size, in twips.	Read/Write	Can be written only when formatting idle.
RightLineStyle	"CRLINEStyle". Gets or sets the right line style.	Read/Write	Can be written only when formatting idle or active.
Suppress	Boolean. Gets or sets the object visibility.	Read/Write	Can be written only when formatting idle or active.
Top	Long. Gets or sets the object upper top position, in twips.	Read/Write	Can be written only when formatting idle or active.
TopCropping	Long. Gets or sets the top cropping size, in twips.	Read/Write	Can be written only when formatting idle.
TopLineStyle	"CRLINEStyle". Gets or sets the top line style.	Read/Write	Can be written only when formatting idle or active.
Width	Long. Gets or sets the object width, in twips.	Read/Write	Can be written only when formatting idle or active.
XScaling	Double. Gets or sets the width scaling factor. For example, 1 means 100%, 2 means 200%, 0.5 means 50% etc. The scaling factor may range from 0.01 to 100.	Read/Write	Can be written only when formatting idle.
YScaling	Double. Gets or sets the height scaling factor. For example, 1 means 100%, 2 means 200%, 0.5 means 50% etc. The scaling factor may range from 0.01 to 100.	Read/Write	Can be written only when formatting idle.

BoxObject Object

The Box Object represents a box that has been drawn on the report. This object allows you to get information about boxes in a report.

BoxObject Object Properties

Examples

“How to format a box object” on page 184

Property	Description	Read/Write	Restriction in event handler
Bottom	Long. Gets or sets the object lower bottom position, in twips.	Read/Write	Can be written only when formatting idle.
BottomRightSection	“Section Object”. Gets the bottom right section.	Read only	Can be written only when formatting idle.
CloseAtPageBreak	Boolean. Gets or sets the close border on page break option.	Read/Write	Can be written only when formatting idle.
ConditionFormula	String. Gets or sets the condition formula. The condition formula can be based on recurring records or on summary fields, but it cannot be based on print-time fields, such as running totals or print-time formulas. Condition formulas cannot have shared variables.	Read/Write	Can be written only when formatting idle.
CornerEllipseHeight	Long. Gets or sets the corner ellipse height, in twips.	Read/Write	
CornerEllipseWidth	Long. Gets or sets the corner ellipse width, in twips.	Read/Write	
CssClass	String. Gets or sets the cascading style sheet Class.	Read/Write	Can be written only when formatting idle.
ExtendToBottomOfSection	Boolean. Gets or sets the extend to bottom of section option.	Read/Write	Can be written only when formatting idle.
FillColor	OLE_COLOR. Gets or sets the fill color.	Read/Write	Can be written only when formatting idle.
HasDropShadow	Boolean. Gets or sets the border drop shadow option..	Read/Write	Can be written only when formatting idle.

Property	Description	Read/Write	Restriction in event handler
Kind	"CRObjektKind". Gets which kind of object (for example, box, cross-tab, field, etc.).	Read only	None
Left	Long. Gets or sets the upper left position, in twips.	Read/Write	Can be written only when formatting idle or active.
LineColor	OLE_COLOR. Gets or sets the line color.	Read/Write	Can be written only when formatting idle.
LineStyle	"CRLLineStyle". Gets or sets the line style.	Read/Write	Can be written only when formatting idle.
LineThickness	Long. Gets or sets the line thickness, in twips.	Read/Write	Can be written only when formatting idle.
Name	String. Gets or sets the object name.	Read/Write	Can be written only when formatting idle.
Parent	"Section Object". Gets reference to the parent object.	Read only	None
Right	Long. Gets or sets the object lower right position, in twips.	Read/Write	Can be written only when formatting idle.
Suppress	Boolean. Gets or sets the object visibility.	Read/Write	Can be written only when formatting idle.
Top	Long. Gets or sets the object upper top position, in twips.	Read/Write	Can be written only when formatting idle.

ConnectionProperty Object

The ConnectionProperty Object is a property bag that stores connection information for the report. Information is dependent on the Crystal Decisions database DLL used by the report and the database drivers that support the DLL. The collection of ConnectionProperty objects can hold common properties defined in our database DLLs, connection properties from the supporting drivers, or both. For example a report based off an OLEDB (ADO) connection to Microsoft SQL Server will contain properties supplied by the database DLL (crdb_ado.dll) such as Provider, Data Source, Initial Catalog, User ID, and Password, as well as the properties supplied by the supporting drivers such as Local Identifier, Connect Timeout and so on. The values for these properties can be set to connect to the current data source for the report or to change the data source for the report.

For more information on the common connection properties see "[Common Connection Properties](#)" on page 30.

ConnectionProperty Object Properties

Examples

“How to connect to a Microsoft Access database” on page 171

“How to connect to a secured Microsoft Access database” on page 173

“How to connect to an ODBC data source” on page 175

“How to connect to an OLEDB data source” on page 177

“How to change the data source” on page 178

“How to list the connection properties and connect to a report” on page 179

Property	Description	Read/Write	Restriction in event handler
ChildProperties	“ ConnectionProperties Collection ”. Gets the child properties of this connection property in a hierarchal property structure.	Read only	None
Description	String. Gets or sets the description of the connection property.	Read/Write	None
LocalizedName	String. Gets or sets the localized name of the connection property.	Read/Write	None
Name	String. Gets or sets the name of the connection property.	Read/Write	None
Value	Variant. Gets or Sets the value of the connection property. Note: Some Values are Write only. For example the value for the Password connection property.	Read/Write	None

Common Connection Properties

The following table lists the connection properties defined in the Crystal Decisions database DLLs. Use these properties when connecting to the desired data source.

Data source	Connection property name	Connection property value description	Remarks
ODBC (RDO) crdb_odbc.dll			
	Connection String	String. The complete connection string containing the driver information. This can be obtained from content of a File DSN.	DSN, FILEDSN and Connection String are mutually exclusive

Data source	Connection property name	Connection property value description	Remarks
	Database	String. The name of the database.	This property can be used together with the DSN, FILEDSN, and Connection String.
	Database Type	String. The type of file. For example: ODBC (RDO) - or - OLE DB (ADO)	This is an optional property.
	DSN	String. The ODBC data source name.	DSN, FILEDSN and Connection String are mutually exclusive.
	FILEDSN	String. Path to the File DSN.	DSN, FILEDSN and Connection String are mutually exclusive
	Password	String. The password used to connect to the data source.	This property can be used together with the DSN, FILEDSN, and Connection String.
	Trusted_Connection	Boolean. Set to True if you want the connection to use NT Authentication.	This property can be used together with the DSN, FILEDSN, and Connection String.
	User ID	String. The user name required to connect to the data source.	This property can be used together with the DSN, FILEDSN, and Connection String.
OLE DB (ADO) crdb_ado.dll			
	Data Source	String. The data source that the OLE DB provider is connected to such as a file path and database name or a server name.	This property can be used together with the Provider and Microsoft Data Link File.
	Database	String. The name of the database.	This property can be used together with the Provider and Microsoft Data Link File.
	Database Type	String. The type of file. For example: ODBC (RDO) - or - OLE DB (ADO)	This is an optional property.

Data source	Connection property name	Connection property value description	Remarks
	Integrated Security	Boolean. Set to True if you want the connection to use NT Authentication.	This property can be used together with the Provider and Microsoft Data Link File.
	Microsoft Data Link File	String. The file path to the Data Link.	Provider and Microsoft Data Link File are mutually exclusive.
	Password	String. The password used to connect to the data source.	This property can be used together with the Provider and Microsoft Data Link File.
	Provider	The name of the OLEDB provider. For example: Microsoft.Jet.OLEDB.4.0 -or- SQLOLEDB	Provider and Microsoft Data Link File are mutually exclusive.
	User ID	String. The user name required to connect to the data source.	This property can be used together with the Provider and Microsoft Data Link File.
Access/Excel (DAO) crdb_dao.dll			
	Database Name	String. The full file path to the database.	This property is required for changing the location and name of the database.
	Database Password	String. The password used to connect to the data source.	This property is only required if the database has database level security.
	Database Type	String. The type of file. For example: ODBC (RDO) - or - OLE DB (ADO)	This is an optional property.
	Session Password	String. The password for a system DSN.	You must provide System Database Path if Session UserID and Session Password are provided.
	Session UserID	String. The user name for a system DSN.	You must provide System Database Path if Session UserID and Session Password are provided.

Data source	Connection property name	Connection property value description	Remarks
	System Database Path	String.	You must provide System Database Path if Session UserID and Session Password are provided.
Oracle crdb_oracle.dll			
	Database Type	String. The type of file. For example: ODBC (RDO) - or - OLE DB (ADO)	This is an optional property.
	Server	String. The Service name.	
	Password	String. The password used to connect to the data source.	
	User ID	String. The user name required to connect to the data source.	
Native connections to local databases. crdb_p2b*.dll			These include Act!, Btrieve, DB2, Dbase, FoxPro, Paradox and so on.
	Database Path	String. The full file path to the database.	This property is required for changing the location and name of the database.
	Database Type	String. The type of file. For example: ODBC (RDO) - or - OLE DB (ADO)	This is an optional property.
	Password	String. The password used to connect to the data source.	
	User ID	String. The user name required to connect to the data source.	
Native connections to server data sources. crdb_p2s*.dll			These include Informix, DB2 Server, Lotus Domino, and so on.
	Database	String. The database name.	

Data source	Connection property name	Connection property value description	Remarks
	Database Type	String. The type of file. For example: ODBC (RDO) - or - OLE DB (ADO)	This is an optional property.
	Server	String. The server name.	
	Password	String. The password used to connect to the data source.	
	User ID	String. The user name required to connect to the data source.	
Field Definitions Only crdb_fielddef.dll			
	Database Type	String. The type of file. For example: ODBC (RDO) - or - OLE DB (ADO)	This is an optional property.
	Field Definition File	String. The full file and path to the Field definition file (*.ttx).	
File System Data crdb_FileSystem.dll			
	Database Type	String. The type of file. For example: ODBC (RDO) - or - OLE DB (ADO)	This is an optional property.
	File Mask	String. A mask to filter the types of files returned. For example: *.exe	
	Include Empty Directories	Boolean. Set to True to include empty directories.	
	Max Number of Subdirectory Levels	Integer. The number of subdirectory levels to include.	Set this value to -1 to include all subdirectories.
	Starting Directory	String. The path to the starting directory.	

ConnectionProperties Collection

The ConnectionProperties Collection is a collection of “[ConnectionProperty Object](#)” in a report. Each object in the collection is a property bag that contains information on a connection property. Use the Item property to access a specific ConnectionProperty Object in the Collection.

For more information on the common connection properties see “[Common Connection Properties](#)” on page 30.

ConnectionProperties Collection Properties

Examples

“[How to connect to a Microsoft Access database](#)” on page 171

“[How to connect to a secured Microsoft Access database](#)” on page 173

“[How to connect to an ODBC data source](#)” on page 175

“[How to connect to an OLEDB data source](#)” on page 177

“[How to change the data source](#)” on page 178

“[How to list the connection properties and connect to a report](#)” on page 179

Property	Definition	Read/Write	Restriction in event handler
Count	Long. Gets the number of objects in the collection.	Read only	None
Item (indexAs Long)	“ ConnectionProperty Object ”. Gets an item from the Collection specified by name.	Read only	None
NameIds	Variant. Returns an array containing the name of each connection property in the collection.	Read only	None

ConnectionProperties Collection Methods

The following methods are discussed in this section:

- “[Add Method \(ConnectionProperties Collection\)](#)” on page 35
- “[Delete Method \(ConnectionProperties Collection\)](#)” on page 36
- “[DeleteAll Method \(ConnectionProperties Collection\)](#)” on page 37

Add Method (ConnectionProperties Collection)

Use the Add method to add a ConnectionProperty Object to the Collection as a name value pair.

Examples

“How to change the data source” on page 178

Syntax

Function Add (Name As String, Value)

Parameter	Description
Name	Specifies the name of the connection property that you want to add to the Collection.
Value	Specifies the value of the connection property.

Remarks

Use this method if alternate connection properties are required to establish a connection to a new data source. For example changing the report’s database driver through the DLLName property in the “DatabaseTable Object”, or connecting to a report created off an OLEDB (ADO) data source to a different provider (Microsoft.Jet.OLEDB.4.0 to SQLOLEDB). Use the “Delete Method (ConnectionProperties Collection)” or the “DeleteAll Method (ConnectionProperties Collection)” to remove any connection properties.

Delete Method (ConnectionProperties Collection)

Use the Delete method to remove a ConnectionProperty Object from the Collection.

Examples

“How to change the data source” on page 178

Syntax

Sub Delete (index)

Parameter

Parameter	Description
index	Specifies the index number of the Object that you want to remove from the Collection.

Remarks

Use this method to remove any connection properties that are no longer required when connecting to a new data source. For example changing the report’s database driver through the DLLName property in the “DatabaseTable Object”, or connecting to a report created off an OLEDB (ADO) data source to a different provider (Microsoft.Jet.OLEDB.4.0 to SQLOLEDB). Use the “Add Method (ConnectionProperties Collection)” to add any new connection properties.

DeleteAll Method (ConnectionProperties Collection)

Use the Delete method to remove all ConnectionProperty Objects from the Collection.

Examples

“How to change the data source” on page 178

Syntax

```
Sub DeleteAll ()
```

Remarks

Use this method to remove all existing connection properties when connecting to a new data source. For example changing the report’s database driver through the DLLName property in the “DatabaseTable Object”, or connecting a report created off an OLEDB (ADO) data source to a different provider (Microsoft.Jet.OLEDB.4.0 to SQLOLEDB). Use the “Add Method (ConnectionProperties Collection)” to add any new connection properties.

CrossTabGroup Object

The CrossTabGroup Object contains information related to CrossTabGroups in a report.

CrossTabGroup Properties

Examples

“How to add, delete and format groups in a crosstab” on page 197

Property	Description	Read/Write	Restriction in event handler
BackColor	OLE_COLOR. Gets or sets the crosstab group background color.	Read/Write	Idle
Condition	“CRGroupCondition”. Gets or sets the crosstab group condition.	Read/Write	Idle
EnableSuppress Label	Boolean. Gets or sets the crosstab group enable suppress label option.	Read/Write	Idle
EnableSuppress Subtotal	Boolean. Gets or sets the crosstab group enable suppress subtotal option.	Read/Write	Idle

Property	Description	Read/Write	Restriction in event handler
Field	A field definition object. Gets or sets the crosstab group's field. See Remarks .	Read/Write	Idle
Parent	"CrossTabObject Object" . Gets reference to the crosstab group parent object.	Read only	None
SortDirection	"CRSortDirection" . Gets or sets the crosstab group sort direction.	Read/Write	Idle

Remarks

After getting the field definition object from the Field property, you can use the Kind property to determine what type of field definition object it is. All field definition objects have the Kind property.

CrossTabGroups Collection

The CrossTabGroups Collection contains CrossTabGroup Objects associated with a report.

CrossTabGroups Collection Properties

Property	Description	Read/Write	Restriction in event handler
Count	Long. Gets the CrossTabGroup count.	Read only	None
Item (index As Long)	"CrossTabGroup Object" . Gets an item from the Collection.	Read only	None
Parent	"CrossTabObject Object" . Gets reference to the parent object.	Read only	None

CrossTabGroups Collection Methods

The following methods are discussed in this section:

- ["Add Method \(CrossTabGroups Collection\)"](#) on page 39
- ["Delete Method \(CrossTabGroups Collection\)"](#) on page 39

Add Method (CrossTabGroups Collection)

Examples

“How to add, delete and format groups in a crosstab” on page 197

Use the Add method to add a field as a CrossTabGroup to the CrossTabGroups Collection.

Syntax

Function Add (Field) As CrossTabGroup

Parameter

Parameter	Description
Field	Specifies the field that you want to add as a CrossTabGroup to the Collection.

Returns

Returns a CrossTabGroup Object member of the Collection.

Delete Method (CrossTabGroups Collection)

Examples

“How to add, delete and format groups in a crosstab” on page 197

Use the Delete method to remove a CrossTabGroup Object from the Collection.

Syntax

Sub Delete (index As Long)

Parameter

Parameter	Description
index	Specifies the index of the CrossTabGroup Object that you want to delete from the Collection.

CrossTabObject Object

The CrossTabObject Object allows you to get and set information for cross-tab objects in a report.

CrossTabObject Object Properties

Property	Description	Read/Write	Restriction in event handler
BackColor	OLE_COLOR. Gets or sets the object background color.	Read/Write	Can be written only when formatting idle or active.
BorderColor	OLE_COLOR. Gets or sets the object border color.	Read/Write	Can be written only when formatting idle or active.
BottomLineStyle	"CRLLineStyle". Gets or sets the bottom line style.	Read/Write	Can be written only when formatting idle or active.
CloseAtPageBreak	Boolean. Gets or sets the close border on page break option.	Read/Write	Can be written only when formatting idle or active.
ConditionFormula	String. Gets or sets the condition formula. The condition formula can be based on recurring records or on summary fields, but it cannot be based on print-time fields, such as running totals or print-time formulas. Condition formulas cannot have shared variables.	Read/Write	Can be written only when formatting idle.
ColumnGrandTotalColor	OLE_COLOR. Gets or sets the column grand total color.	Read/Write	Idle
ColumnGroups	"Remarks". Gets the column groups Collection.	Read only	None
CssClass	String. Gets or sets the cascading style sheet Class.	Read/Write	Can be written only when formatting idle.
EnableKeepColumnsTogether	Boolean. Gets or sets the enable keep columns together option.	Read/Write	Idle
EnableRepeatRowLabels	Boolean. Gets or sets the enable repeat row labels option.	Read/Write	Idle
EnableShowCellMargins	Boolean. Gets or sets the enable show cell margins option.	Read/Write	Idle
EnableShowGrid	Boolean. Gets or sets the enable show grid option.	Read/Write	Idle

Property	Description	Read/Write	Restriction in event handler
EnableSuppressColumnGrandTotals	Boolean. Gets or sets the enable suppress column grand totals option.	Read/Write	Idle
EnableSuppressEmptyColumns	Boolean. Gets or sets the enable suppress empty columns option.	Read/Write	Idle
EnableSuppressEmptyRows	Boolean. Gets or sets the enable suppress empty rows option.	Read/Write	Idle
EnableSuppressRowGrandTotals	Boolean. Gets or sets the enable suppress row grand totals option.	Read/Write	Idle
HasDropShadow	Boolean. Gets or sets the border drop shadow option.	Read/Write	Can be written only when formatting idle or active.
Height	Long. Gets the object height, in twips.	Read only	Can be written only when formatting idle or active.
KeepTogether	Boolean. Gets or sets the keep object together option.	Read/Write	Can be written only when formatting idle or active.
Kind	"CROBJECTKind". Gets which kind of object (for example, box, cross-tab, field, etc.).	Read only	None
Left	Long. Gets or sets the upper left position, in twips.	Read/Write	Can be written only when formatting idle or active.
LeftLineStyle	"CRLINEStyle". Gets or sets the left line style.	Read/Write	Can be written only when formatting idle or active.
Name	String. Gets or sets the object name.	Read/Write	Can be written only when formatting idle.
Parent	"Section Object". Gets reference to the parent object.	Read only	None
RightLineStyle	"CRLINEStyle". Gets or sets the right line style.	Read/Write	Can be written only when formatting idle or active.
RowGrandTotalColor	OLE_COLOR. Gets or sets the row grand total color.	Read/Write	Idle
RowGroups	"Remarks". Gets the row groups Collection.	Read Only	None
SummaryFields	"ObjectSummaryFieldDefinitions Collection". Gets the summary fields.	Read Only	None
SpecifiedGroups	Gets or sets the specified groups from a row or column in the crosstab.	Read/Write	Can be written only when formatting idle

Property	Description	Read/Write	Restriction in event handler
Suppress	Boolean. Gets or sets the object visibility option.	Read/Write	Can be written only when formatting idle or active.
Top	Long. Gets or sets the object upper top position, in twips.	Read/Write	Can be written only when formatting idle or active.
TopLineStyle	<i>"CRLLineStyle"</i> . Gets or sets top line style.	Read/Write	Can be written only when formatting idle or active.
Width	Long. Gets the object width, in twips.	Read Only	Can be written only when formatting idle or active.

Database Object

The Database Object provides properties to get information about the database accessed by a report. See *"Overview of the Report Designer Object Model"* on page 14.

Database Object Properties

Property	Description	Read/Write	Restriction in event handler
ConnectionProperties	<i>"ConnectionProperties Collection"</i> . Gets the collection of connection properties from the table.	Read only	None.
Links	<i>"TableLinks Collection"</i> . Gets database link collection.	Read only	None
Parent	<i>"Report Object"</i> . Gets reference to the parent object.	Read only	None
Tables	<i>"DatabaseTables Collection"</i> . Gets the DatabaseTables Collection which specifies the database objects used in the report (for example, an Access report may contain a query or a SQL Server report may be based on a stored procedure - if so, they will be returned as part of this collection along with the database table used in the report).	Read only	None

Database Object Methods

The following methods are discussed in this section:

- “AddADOCommand Method (Database Object)” on page 43
- “AddOLEDBSource Method (Database Object)” on page 43
- “LogOffServer Method (Database Object)” on page 44
- “LogOnServer Method (Database Object)” on page 45
- “LogOnServerEx Method (Application Object)” on page 20
- “SetDataSource Method (Database Object)” on page 47
- “Verify Method (Database Object)” on page 47

AddADOCommand Method (Database Object)

Use the AddADOCommand method to add a database table to your report through an ADO connection and command.

Syntax

```
Sub AddADOCommand (pConnection, pCommand)
```

Parameters

Parameter	Description
pConnection	Specifies the ADO connection that you want to use.
pCommand	Specifies the ADO command that you want to use.

AddOLEDBSource Method (Database Object)

Use the AddOLEDBSource method to add a database table to your report through an OLE DB provider.

Syntax

```
Sub AddOLEDBSource (pConnectionString As String, pTableName As String)
```

Parameters

Parameter	Description
pConnectionString	Specifies the connection string for the OLE DB provider.
pTableName	Specifies the OLE DB database table that you want to add to your report.

Remarks

Some databases such as Oracle are case sensitive and require that proper case be applied to the database names and table names in the connection string.

```

'Instantiate the report object.
Set m_Report = New CrystalReport1

' Convert the database driver to ODBC.
m_Report.Database.ConvertDatabaseDriver "p2sodbc.dll", True

' Set the logon information for the ODBC data source.
' If the logon information is not set, an error will be produced when the
' report is previewed or exported.
m_Report.Database.Tables(1).SetLogOnInfo "Xtreme Sample Database", "", "",
""

' Verify the database.
' If the database is not verified before exporting, an error will be
produced.
' If the database is not verified before previewing the report, the user may
be
' prompted when the report is refreshed in the Crystal Report Viewer.
m_Report.Database.Verify

```

LogOffServer Method (Database Object)

The LogOffServer method logs off an SQL server, ODBC or other data source. Use this method when you have logged on to the data source using LogOnServer. This method can be invoked only in formatting Idle mode.

Syntax

```

Sub LogOffServer (pDllName As String, pServerName As String,
    [pDatabaseName], [pUserID], [pPassword])

```

Parameters

Parameter	Description
pDllName	Specifies the name of the DLL for the server or password protected non-SQL table you want to log on to, for example "PDSODBC.DLL". Note that the dllName must be enclosed in quotes. DLL names have the following naming convention: PDB*.DLL for standard (non-SQL) databases, PDS*.DLL for SQL/ODBC databases.
pServerName	Specifies the log on name for the server used to create the report. For ODBC, use the data source name. This value is case-sensitive. See Remarks below.
[pDatabaseName]	Specifies the name for the database used to create the report. See Remarks below.
[pUserID]	Specifies the User ID number necessary to log on to the server. See Remarks below.
[pPassword]	Specifies the password necessary to log on to the server.

Remarks

- When you pass an empty string (""), for pServerName, pDatabaseName, or pUserID, the program uses the value that's already set in the report. If you want to override a value that's already set in the report, use a non-empty string (for example, "Server A").
- If you try to log off a server that is still in use (that is, there is an object variable still in focus that holds reference to a report that requires being logged on to the server to access data) you will be unable to log off. This will apply to every object that comes from the “Report Object”, as they all hold reference to the report through their respective Report properties.
- If you assign the Report object to the ReportSource property of the “CRViewer Object (CRVIEWERLib)”, in the Crystal Reports Report Viewer, enabling the report to be displayed through the Report Viewer, you cannot call LogOffServer for the report until you assign a new report to the Report Viewer or close the CRViewer object.

LogOnServer Method (Database Object)

The LogOnServer method logs on to an SQL server, ODBC or other data source. Once logged on using this method, you will remain logged on until you call LogOffServer or until the Application Object is destroyed and craxdrt.dll is unloaded. This method can be invoked only in formatting Idle mode.

Syntax

```
Sub LogOnServer (pDllName As String, pServerName As String,  
                [pDatabaseName], [pUserID], [pPassword])
```

Parameters

Parameter	Description
pDllName	Specifies the name of the DLL for the server or password protected non-SQL table you want to log on to, for example "PDSODBC.DLL". Note that the dllName must be enclosed in quotes. DLL names have the following naming convention: PDB*.DLL for standard (non-SQL) databases, PDS*.DLL for SQL/ODBC databases.
pServerName	Specifies the log on name for the server used to create the report. For ODBC, use the data source name. This value is case-sensitive. See Remarks below.
[pDatabaseName]	Specifies the name for the database used to create the report. See Remarks below.
[pUserID]	Specifies the User ID number necessary to log on to the server. See Remarks below.
[pPassword]	Specifies the password necessary to log on to the server.

Remarks

When you pass an empty string ("") for this parameter, the program uses the value that's already set in the report. If you want to override a value that's already set in the report, use a non-empty string (for example, "Server A").

LogOnServerEx Method (Database Object)

The LogOnServerEx method logs on to an SQL server, ODBC or other data source. Using this method, you can pass ServerType and ConnectionString info. Once logged on using this method, you will remain logged on until you call LogOffServer or until the Application Object is destroyed and craxdr.dll is unloaded. This method can be invoked only in formatting Idle mode.

Syntax

```
Sub LogOnServerEx (pDllName As String, pServerName As String,  
    [pDatabaseName], [pUserID], [pPassword],  
    [pServerType], [pConnectionString])
```

Parameters

Parameter	Description
pDllName	Specifies the name of the DLL for the server or password protected non-SQL table you want to log on to, for example "PDSODBC.DLL". Note that the dllName must be enclosed in quotes. DLL names have the following naming convention: PDB*.DLL for standard (non-SQL) databases, PDS*.DLL for SQL/ODBC databases.
pServerName	Specifies the log on name for the server used to create the report. For ODBC, use the data source name. This value is case-sensitive. See Remarks below.
[pDatabaseName]	Specifies the name for the database used to create the report. See Remarks below.
[pUserID]	Specifies the User ID number necessary to log on to the server. See Remarks below.
[pPassword]	Specifies the password necessary to log on to the server.
[pServerType]	Specifies the database Server Type.
[pConnectionString]	Specifies the connection string.

Remarks

When you pass an empty string ("") for this parameter, the program uses the value that's already set in the report. If you want to override a value that's already set in the report, use a non-empty string (for example, "Server A").

SetDataSource Method (Database Object)

The SetDataSource method is used to provide information about a data source to the database driver associated with this Database object at runtime. For instance, if a report has been designed using the Crystal Active Data Driver this method can be used to provide an active data source for the report, such as a DAO, ADO, or RDO Recordset or a CDO Rowset. In this case, the object passed to the second parameter of this method replaces, at runtime, the field definition file used to create the report. This method can be invoked only in formatting Idle mode. When using a secure connection such as SQL Server, some additional code is required (see Remarks below).

Syntax

```
Sub SetDataSource (data, [dataTag], [tableNumber])
```

Parameters

Parameter	Description
data	Variant data passed to the database driver. For example, with Active data, this must be a Recordset object if you are using DAO, ADO, or the Visual Basic data control. This must be a Rowset object if you are using CDO.
[dataTag]	A value indicating the type of data being passed to the DatabaseTable object in the Data parameter. Currently, the only possible value is 3. This value must be used for all Active data sources including DAO, ADO, RDO, CDO, and the Visual Basic data control.
[tableNumber]	Specifies the index number of the table to be set. Default value = 1.

Remarks

- When the data source uses a secure connection, such as SQL Server, additional information must be passed in the "form load" event before the call to view the report. For example,

```
DataEnvironment1.Command1  
Report.Database.SetDataSource (DataEnvironment1.rsCommand1)
```

- SetDataSource method is used to set a datasource at runtime. If the report is initially created and then saved, and then later run using either the RDC or CRW, the runtime datasource (DAO, ADO, or RDO Recordset) cannot be recreated. The user will not be able to run or preview the report.

Verify Method (Database Object)

The Verify method verifies that the location of the database is still valid and checks to see if any changes have been made to table design, etc. If there are any changes to the database, the Verify method will update the report automatically to reflect these changes. See Remarks below. This method can be invoked only in formatting Idle mode.

Syntax

```
Sub Verify ()
```

Remarks

Prior to calling Verify, you can use the “[CheckDifferences Method \(DatabaseTable Object\)](#)” to determine what kind of differences, if any, exist between the report table and the physical table. pDifferences parameter of CheckDifferences method will pass back one or more bitwise (XOR'd) “[CRTableDifferences](#)” enums indicating the information that you want to retrieve.

DatabaseFieldDefinition Object

The DatabaseFieldDefinition Object represents a database field used in the report. This object provides properties for getting information on database fields in the report.

DatabaseFieldDefinition Object Properties

Property	Definition	Read/Write	Restriction in event handler
DatabaseFieldDisplayName	String. Gets the display name for the field in the database.	Read Only	None
DatabaseFieldName	String. Gets the name of the field in the database (for example, Product ID).	Read only	None
Kind	“ CRFieldKind ”. Gets which kind of field (for example, database, summary, formula, etc.).	Read only	None
Name	String. Gets the unique Crystal formula form name of the field within the report as {table.FIELD} (for example, {product.PRODUCT ID}).	Read only	None
NextValue	Variant. Gets the field next value.	Read only	Can be read only when top-level Report object is formatting active.
NumberOfBytes	Integer. Gets the number of bytes required to store the field data in memory.	Read only	None
Parent	“ Report Object ”. Gets reference to the parent object.	Read only	None

Property	Definition	Read/Write	Restriction in event handler
PreviousValue	Variant. Gets the field previous value.	Read only	Can be read only when top-level Report object is formatting active.
TableAliasName	String. Gets the alias for the table containing the field.	Read only	None
Value	Variant. Gets the field current value.	Read only	Can be read only when top-level Report object is formatting active.
ValueType	"CRFieldValueType". Gets which type of value is found in the field.	Read only	None

DatabaseFieldDefinitions Collection

The DatabaseFieldDefinitions Collection is a collection of database field definition objects. One object exists in the collection for every database field accessed by the report. Access a specific "DatabaseFieldDefinition Object" in the collection using the Item property.

DatabaseFieldDefinitions Collection Properties

Property	Description	Read/Write	Restriction in event handler
Count	Long. Gets the number of "DatabaseFieldDefinition Object", in the collection.	Read only	None
Item (index As Long)	"DatabaseFieldDefinition Object". Item has an index parameter that is a numeric, 1-based index for the object that you want to retrieve (for example, Item (1) for the first database field in the collection). The items in the collection are indexed in the order they appear in the database table.	Read only	None
Parent	"DatabaseTable Object". Reference to the parent object.	Read only	None

Remarks

Instead of using the Item property as shown, you can reference a database directly (for example, DatabaseFieldDefinitions(1)).

DatabaseFieldDefinitions Collection Methods

The following method is discussed in this section:

- [“GetItemByName Method \(DatabaseFieldDefinitions Collection\)”](#) on page 50

GetItemByName Method (DatabaseFieldDefinitions Collection)

Use the GetItemByName method to retrieve a DatabaseFieldDefinition object by name.

Syntax

```
Sub GetItemByName(name As String, databaseFieldDefinition)
```

Parameters

Parameter	Description
name	The item’s unique name.
databaseFieldDefinition	“DatabaseFieldDefinition Object” . An object to hold the retrieved item.

DatabaseTable Object

The DatabaseTable Object refers to a database table accessed by the report.

DatabaseTable Object Properties

Examples

[“How to connect to a Microsoft Access database”](#) on page 171

[“How to connect to a secured Microsoft Access database”](#) on page 173

[“How to connect to an ODBC data source”](#) on page 175

[“How to connect to an OLEDB data source”](#) on page 177

[“How to change the data source”](#) on page 178

Property	Description	Read/Write	Restriction in event handler
ConnectBufferString	String. Gets the table connect buffer string.	Read only	None
DatabaseType	“CRDatabaseType” . Gets the database table type	Read only	None
DescriptiveName	String. Gets the table descriptive name.	Read only	None

Property	Description	Read/Write	Restriction in event handler
DllName	String. Gets the table driver DLL name.	Read only	None
Fields	"DatabaseFieldDefinitions Collection Properties". Gets the collection of database fields in the table.	Read only	None
Location	String. Gets or sets the location of the database table.	Read/Write	Can be written only when formatting idle.
Name	String. Gets or sets the alias name for the database table used in the report.	Read/Write	Can be written only when formatting idle.
Parent	"Database Object". Reference to the parent object.	Read only	None

DatabaseTable Object Methods

The following methods are discussed in this section:

- "CheckDifferences Method (DatabaseTable Object)" on page 51
- "SetDataSource Method (DatabaseTable Object)" on page 51
- "SetTableLocation Method (DatabaseTable Object)" on page 52
- "TestConnectivity Method (DatabaseTable Object)" on page 53

CheckDifferences Method (DatabaseTable Object)

Use the CheckDifferences method to determine what kind(s) of differences were found between the report table and the physical table.

Syntax

Sub CheckDifferences (pDifferences As Long, [reserved])

Parameters

Parameter	Description
pDifferences	"CRTableDifferences". Bitwise constants specify the table difference(s) (XOR'd), if any.
[reserved]	Reserved. Do not use.

SetDataSource Method (DatabaseTable Object)

The SetDataSource method is used to provide information about a data source to the database driver associated with this DatabaseTable object at runtime. For instance, if a report has been designed using the Crystal Active Data Driver this method can be used to provide an active data source for the report, such as a DAO, ADO, or RDO Recordset or a CDO Rowset. In this case, the object passed to the

second parameter of this method replaces, at runtime, the field definition file used to create the report. This method can be invoked only in formatting Idle mode. When using a secure connection such as SQL Server, some additional code is required (see Remarks below).

Syntax

```
Sub SetDataSource (data, [dataTag])
```

Parameters

Parameter	Description
data	Variant data passed to the database driver. For example, with Active data, this must be a Recordset object if you are using DAO, ADO, or the Visual Basic data control. This must be a Rowset object if you are using CDO.
[dataTag]	A value indicating the type of data being passed to the DatabaseTable object in the Data parameter. Currently, the only possible value is 3. This value must be used for all Active data sources including DAO, ADO, RDO, CDO, and the Visual Basic data control.

Remarks

- When the data source uses a secure connection, such as SQL Server, additional information must be passed in the "form load" event before the call to view the report. For example,


```
DataEnvironment1.Command1
Report.DatabaseTable.SetDataSource (DataEnvironment1.rsCommand1)
```
- SetDataSource method is used to set a datasource at runtime. If the report is initially created and then saved, and then later run using either the RDC or CRW, the runtime datasource (DAO, ADO, or RDO Recordset) cannot be recreated. The user will not be able to run or preview the report.

SetTableLocation Method (DatabaseTable Object)

The SetTableLocation method is used to set the DatabaseTable location, sublocation, and connect buffer string.

Examples

["How to connect to a Microsoft Access database" on page 171](#)

["How to connect to a secured Microsoft Access database" on page 173](#)

Syntax

```
Sub SetTableLocation (pLocation As String, pSubLocation As String,
    pConnectBufferSting As String)
```

Parameters

Parameter	Description
pLocation	Specifies the location of the database table (file path and name.ext).
pSubLocation	Specifies the sublocation of the database table.
pConnectBufferSting	Specifies the connection buffer string.

Remarks

For example:

```
object.SetTableLocation "xtreme.mdb", "Customer", ""
```

TestConnectivity Method (DatabaseTable Object)

The TestConnectivity method tests to see if the database can be logged on to with the current information and if the database table can be accessed by the report.

Syntax

```
Function TestConnectivity () As Boolean
```

Returns

- TRUE if the database session, log on, and location information is all correct.
- FALSE if the connection fails or an error occurs.

DatabaseTables Collection

The DatabaseTables Collection is a collection of DatabaseTable objects. A DatabaseTable object exists for every database object (for example, table, query, stored procedure, etc.) accessed by the report. Access a specific DatabaseTable Object in the collection using the Item property.

DatabaseTables Collection Properties

Property	Description	Read/Write	Restriction in event handler
Count	Long. Gets the number of database objects in the collection.	Read only	None
Item (indexAs Long)	“DatabaseTable Object”. Item has an index parameter that is a numeric, 1-based index (for example, Item (1)). The items in the collection are indexed in the order in which they were added to the report.	Read only	None
Parent	“Database Object”. Reference to the parent object.	Read only	None

Remarks

Instead of using the Item property as shown, you can reference a table directly (for example, DatabaseTable(1)).

DatabaseTables Collection Methods

The following methods are discussed in this section:

- “Add Method (DatabaseTables Collection)” on page 54
- “AddStoredProcedure Method (DatabaseTables Collection)” on page 55
- “Delete Method (DatabaseTables Collection)” on page 56

Add Method (DatabaseTables Collection)

Use the Add method to add a database table to the report.

Syntax

```
Sub Add (pLocation As String, [pSubLocation], [pConnectInfo],  
        [tableType], [pDllName], [pServerName], [pServerType],  
        [pDatabaseName], [pUserID], [pPassword])
```

Parameters

Parameter	Description
pLocation	Specifies the location of the database table that you want to add to the report.
[pSubLocation]	Specifies the sublocation of the database table that you want to add to the report.
[pConnectInfo]	Specifies the connection string.
[tableType]	Specifies the type of database table that you want to add to the report.
[pDllName]	Specifies the DLL name for the database containing the table that you want to add.
[pServerName]	Specifies the database Server Name.
[pServerType]	Specifies the database Server Type.
[pDatabaseName]	Specifies the database (file path and name.ext) containing the table that you want to add.
[pUserID]	Specifies the User's ID.
[pPassword]	Specifies the User's Password.

Remarks

- DatabaseTables.Add method is very generic and can be used to add tables to a report from all kinds of data sources (for example, PC Table, SQL Server, ODBC, OLE DB provider, ADO, RDO, DAO Recordset).

- For example:

```
object.Add "xtreme.mdb", "Customer"
```
- You may use Add() to add a stored procedure to a report. No initial values are passed to the procedure's parameters so you need to set the values later using other API functions. To pass parameter values at the time of adding the stored procedure, use the "AddStoredProcedure Method (DatabaseTables Collection)" instead.

AddStoredProcedure Method (DatabaseTables Collection)

Use the AddStoredProcedure method to add a stored procedure and its initial parameter values to the report.

Syntax

```
Sub AddStoredProcedure (pLocation as String,  
    [pSubLocation],[pConnectionInfo],[tableType],[pDllName],  
    [pServerType],[pDatabaseName],[pUserID],[pPassword],[paramValueInfos])
```

Parameters

Parameter	Description
pLocation	Specifies the location of the database table that you want to add to the report.
[pSubLocation]	Specifies the sublocation of the database table that you want to add to the report.
[pConnectInfo]	Specifies the connection string.
[tableType]	Specifies the type of database table that you want to add to the report.
[pDllName]	Specifies the DLL name for the database containing the table that you want to add.
[pServerName]	Specifies the database Server Name.
[pServerType]	Specifies the database Server Type.
[pDatabaseName]	Specifies the database (file path and name.ext) containing the table that you want to add.
[pUserID]	Specifies the User's ID.
[pPassword]	Specifies the User's Password.
[ParamValuesInfos]	ParameterValueInfos Collection. Specifies the initial values for the stored procedure's parameters.

Remarks

AddStoredProcedure() has one additional parameter that Add() does not have: ParamValuesInfos. This parameter allows the initial values for the stored procedure's parameters to be passed at the same time as the procedure is added to the report. If you do not want to pass initial parameter values, use "Add Method (DatabaseTables Collection)" instead.

Delete Method (DatabaseTables Collection)

Use the Delete method to remove a database table from the Collection.

Syntax

Sub Delete (index As Long)

Parameter

Parameter	Description
index	Specifies the 1-based index number in the Collection of the database table that you want to delete.

ExportOptions Object

The ExportOptions object provides properties and methods for retrieving information and setting options for exporting your report (that is, export format, destination, etc.). An ExportOptions Object is obtained from the ExportOptions property of the *“Report Object”*.

ExportOptions Object Properties

Property	Description	Read/Write	Restriction in event handler
ApplicationFile Name	String. Gets or sets the destination application file name.	Read/Write	Set before exporting report to an application destination.
CharFieldDelimiter	String. Gets or sets the character used to separate fields in character separated text formats. This character delimits every field in the file.	Read/Write	None
CharString Delimiter	String. Gets or sets the character used to separate strings in character separated text formats. This character delimits only string fields (numeric, date fields, etc., have no delimiter).	Read/Write	None
DestinationDLL Name	String. Gets the Internal Name property of the DLL used to export the report to a certain destination. The destination is set in the DestinationType property.	Read only	None
DestinationType	<i>“CRExportDestinationType”</i> . Gets or sets the export destination type.	Read/Write	None

Property	Description	Read/Write	Restriction in event handler
DiskFileName	String. Gets or sets the file name if the report is exported to a disk. When exporting to HTML use HTMLFileName. When exporting to XML use XMLFileName.	Read/Write	None
ExcelAreaGroup Number	Integer. Gets or sets the base area group number if the area type is group area when exporting to Excel.	Read/Write	None
ExcelAreaType	" CRAreaKind ". Gets or sets the base area type if not using constant column width when exporting to Excel.	Read/Write	None
ExcelConstant ColumnWidth	Double. Gets or sets the column width when exporting to Excel.	Read/Write	None
ExcelConvertDate ToString	Boolean. Gets or sets export to Excel has converting date values to strings option.	Read/Write	None
ExcelExportAll Pages	Boolean. Gets or sets export to Excel with all pages.	Read/Write	None
ExcelFirstPage Number	Long. Gets or sets export to Excel for first page number.	Read/Write	None
ExcelLastPage Number	Long. Gets or sets export to Excel for last page number.	Read/Write	None
ExcelPageBreaks	Boolean. Gets or sets export to Excel has page break option.	Read/Write	None
ExcelTabHas ColumnHeadings	Boolean. Gets or sets exporting to Excel has column headings option.	Read/Write	None
ExcelUseConstant ColumnWidth	Boolean. Gets or sets export to Excel to use constant column width.	Read/Write	None
ExcelUseTabular Format	Boolean. Gets or sets exporting to Excel to use tabular format.	Read/Write	None
ExcelUseWorksheet Functions	Boolean. Gets or sets export to Excel to use worksheet functions to represent subtotal fields in the report.	Read/Write	None
Exchange DestinationType	" CRExchangeDestinationType ". Gets or sets the Exchange destination type for reports exported to Exchange folders.	Read/Write	None
ExchangeFolder Path	String. Gets or sets the path of the Exchange folder for reports exported to Exchange (for example, "MyFolders@Inbox").	Read/Write	None
ExchangePassword	String. Sets Exchange password.	Write only	None

Property	Description	Read/ Write	Restriction in event handler
ExchangeProfile	String. Gets or sets a user profile for accessing an Exchange folder for reports exported to Exchange.	Read/ Write	None
ExchangePathHas ColumnHeadings	Boolean. Gets or sets the column heading option when exporting to Exchange.	Read/ Write	None
FormatDLLName	String. Gets the Internal Name property of the DLL used to export the report to a certain format type. The export format type is set in the FormatType property.	Read only	None
FormatType	<i>"CRExportFormatType"</i> . Gets or sets the format type for the exported report (for example, text, Excel, etc.).	Read/ Write	None
HTMLEnable SeperatedPages	Boolean. Gets or sets the option to create seperated pages when exporting to HTML format.	Read/ Write	None
HTMLFileName	String. Gets or sets the HTML file name for reports exported to HTML format.	Read/ Write	None
HTMLHasPage Navigator	Boolean. Gets or sets the option to display a page navigator on each page of a report exported to HTML format.	Read/ Write	None
LotusDomino Comments	String. Gets or sets the destination Lotus Domino comments.	Read/ Write	None
LotusDomino DatabaseName	String. Gets or sets the destination Lotus Domino database name.	Read/ Write	None
LotusNotesForm Name	String. Gets or sets the destination Lotus Domino form name.	Read/ Write	None
MailBccList	String. Gets or sets a Blind Carbon Copy (BCC) list for reports e-mailed to a VIM e-mail account.	Read/ Write	None
MailCcList	String. Gets or sets a Carbon Copy (CC) list for reports e-mailed.	Read/ Write	None
MailMessage	String. Gets or sets the e-mail message included with e-mailed reports.	Read/ Write	None
MailSubject	String. Gets or sets the e-mail subject heading for reports being e-mailed.	Read/ Write	None
MailToList	String. Gets or sets the To list for reports being e-mailed.	Read/ Write	None
MailUserName	String. Gets or sets mail user name.	Read/ Write	None

Property	Description	Read/Write	Restriction in event handler
NumberOfLinesPerPage	Integer. Gets or sets the number of lines to appear per page option for reports exported to a paginated text format.	Read/Write	None
ODBCDataSourceName	String. Gets or sets the ODBC data source for reports exported to ODBC.	Read/Write	None
ODBCDataSourcePassword	String. Sets the ODBC data source password.	Write only	None
ODBCDataSourceUserID	String. Gets or sets the user name used to access an ODBC data source for reports exported to ODBC.	Read/Write	None
ODBCExportTableName	String. Gets or sets the database table in the ODBC data source that the report file exported to ODBC will be appended to. You can also create a new table using this property.	Read/Write	None
Parent	"Report Object". Reference to the parent object.	Read only	None
PDFExportAllPages	Boolean. Gets or sets whether or not to export all pages of the report to Portable Document Format(PDF). PDFExportAllPages must be set to false when setting PDFFirstPageNumber and PDFLastPageNumber.	Read/Write	None
PDFFirstPageNumber	Long. Gets or sets the start page, of a page export range, when exporting to PDF. PDFExportAllPages must be set to False or this value is ignored.	Read/Write	None
PDFLastPageNumber	Long. Gets or sets the end page, of a page export range, when exporting to PDF. PDFExportAllPages must be set to False or this value is ignored.	Read/Write	None
RTFExportAllPages	Boolean. Gets or sets whether or not to export all pages of the report to Rich Text Format(RTF). RTFExportAllPages must be set to false when setting RTFFirstPageNumber and RTFLastPageNumber.	Read/Write	None
RTFFirstPageNumber	Long. Gets or sets the start page, of a page export range, when exporting to RTF. RTFExportAllPages must be set to False or this value is ignored.	Read/Write	None

Property	Description	Read/Write	Restriction in event handler
RTFFLastPageNumber	Long. Gets or sets the end page, of a page export range, when exporting to RTF. RTFExportAllPages must be set to False or this value is ignored.	Read/Write	None
UseDefaultCharactersPerInch	Boolean. Gets or sets use default characters per inch option.	Read/Write	None
UserDefinedCharactersPerInch	Long. Gets or sets user defined characters per inch.	Read/Write	None
UseReportDateFormat	Boolean. Gets or sets whether the date format used in the report should also be used in the exported report. Can be used for Data Interchange Format (DIF), Record Style Format, and comma, tab, or character separated format.	Read/Write	None
UseReportNumberFormat	Boolean. Gets or sets whether the number format used in the report should also be used in the exported report. Can be used for Data Interchange Format (DIF), Record Style Format, and comma, tab, or character separated format.	Read/Write	None
XMLAllowMultipleFiles	Boolean. Gets or sets allow multiple files, when exporting to XML. When set to True the Schema file for the report will be exported along with the XML file. The Schema file will be either an XML schema (.xsd) or a Document Type Definition (.dtd), depending on the options selected in the XML Forma dilaog box. For more information see “Customizing XML report definitions” in the <i>Crystal Reports User’s Guide</i>	Read/Write	None
XMLFileName	String. Gets or sets the file name if the report is exported to a disk.	Read/Write	None

Remarks

For backwards compatibility the FormatDllName and DestinationDllName properties return the Internal Name property of the associated DLL. The Internal Name property of the DLL is found in the DLLs Properties Dialog box under the Version tab. For a list of export DLLs see “Export Destination” and “Export Format” in the *Runtime help (Runtime.hlp)*.

ExportOptions Object Methods

The following methods are discussed in this section:

- “PromptForExportOptions Method (ExportOptions Object)” on page 61
- “Reset Method (ExportOptions Object)” on page 61

PromptForExportOptions Method (ExportOptions Object)

The PromptForExportOptions method prompts the user for export information using default Crystal Report Engine dialog boxes.

Syntax

```
Sub PromptForExportOptions ()
```

Reset Method (ExportOptions Object)

The Reset method clears all ExportOptions properties.

Syntax

```
Sub Reset ()
```

FieldDefinitions Collection

The FieldDefinitions Collection contains the various types of XXXFieldDefinition Objects (for example, DatabaseFieldDefinition, FormulaFieldDefinition, SummaryFieldDefinition). For the current release, developers can access the FieldDefinitions Collection only through the ConditionFields Property of GraphObject.

FieldDefinitions Collection Properties

Property	Description	Read/Write	Restriction in event handler
Count	Long. Gets the collection count.	Read only	None
Item (index As Long)	A field definition object. Gets the collection item. See Remarks .	Read only	None
Parent	“Report Object”. Reference to the parent object.	Read only	None

Remarks

After getting the field definition object from the Item property, you can use the Kind property to determine what type of field definition object the item is. All field definition objects have the Kind property.

FieldDefinitions Collection Methods

The following methods are discussed in this section:

- “Add Method (FieldDefinitions Collection)” on page 62
- “Delete Method (FieldDefinitions Collection)” on page 62

Add Method (FieldDefinitions Collection)

Use the Add method to add the specified Field to the FieldDefinitions Collection.

Syntax

```
Sub Add (Field)
```

Parameter

Parameter	Description
Field	Specifies the Field that you want to add to the Collection.

Delete Method (FieldDefinitions Collection)

Use the Delete method to remove the specified Field from the FieldDefinitions Collection.

Syntax

```
Sub Delete (Field)
```

Parameter

Parameter	Description
Field	Specifies the Field that you want to delete from the Collection.

FieldElement Object

The FieldElement Object represents a field object embedded in a text field. This object provides properties for retrieving field element information from a text object (for example, line spacing, the number of decimal places, font color). A FieldElement Object is obtained from the Item property of the “FieldElements Collection” on page 66.

FieldElement Object Properties

Property	Description	Read/Write	Restriction in event handler
AmPmType	" CRAMPmType ". Gets or sets the AM/PM type option.	Read/Write	Can be written only when formatting idle or active.
AmString	String. Gets or sets the AM string.	Read/Write	Can be written only when formatting idle or active.
BooleanOutputType	" CRBooleanOutputType ". Gets or sets the Boolean output type.	Read/Write	Can be written only when formatting idle or active.
CharacterSpacing	Long. Gets or sets the character spacing. OLE_COLOR. Gets or sets the object background color.	Read/Write	Can be written only when formatting idle.
Color	OLE_COLOR. Gets or sets the font color.	Read/Write	Can be written only when formatting idle or active.
ConditionFormula	String. Gets or sets the condition formula. The condition formula can be based on recurring records or on summary fields, but it cannot be based on print-time fields, such as running totals or print-time formulas. Condition formulas cannot have shared variables.	Read/Write	Can be written only when formatting idle.
CurrencyPositionType	" CRCurrencyPositionType ". Gets or sets the currency position type.	Read/Write	Can be written only when formatting idle or active.
CurrencySymbol	String. Gets or sets the currency symbol.	Read/Write	Can be written only when formatting idle or active.
CurrencySymbolType	" CRCurrencySymbolType ". Gets or sets the currency symbol type.	Read/Write	Can be written only when formatting idle or active.
DateCalendarType	" CRDateCalendarType ". Gets or sets the date calendar type.	Read/Write	Can be written only when formatting idle.
DateEraType	" CRDateEraType ". Gets or sets the date era type.	Read/Write	Can be written only when formatting idle.
DateFirstSeparator	String. Gets or sets the date first separator.	Read/Write	Can be written only when formatting idle or active.
DateOrder	" CRDateOrder ". Gets or sets the date order.	Read/Write	Can be written only when formatting idle or active.
DatePrefixSeparator	String. Gets or sets the date prefix separator.	Read/Write	Can be written only when formatting idle or active.
DateSecondSeparator	String. Gets or sets the date second separator.	Read/Write	Can be written only when formatting idle or active.
DateSuffixSeparator	String. Gets or sets the date suffix separator	Read/Write	Can be written only when formatting idle or active.

Property	Description	Read/Write	Restriction in event handler
DateWindowsDefaultType	" CRDateTimeFieldFormatConditionFormulaType ". Gets or sets the date windows default type.	Read/Write	Can be written only when formatting idle.
DayType	" CRDayType ". Gets or sets the day type.	Read/Write	Can be written only when formatting idle or active.
DecimalPlaces	Integer. Gets or sets the number decimal places.	Read/Write	Can be written only when formatting idle or active.
DecimalSymbol	String. Gets or sets the decimal symbol.	Read/Write	Can be written only when formatting idle or active.
DisplayReverseSign	Boolean. Gets or sets the reverse sign option.	Read/Write	Can be written only when formatting idle or active.
FieldDefinition	A field definition object. Gets the field definition. See Remarks .	Read Only	None
FirstLineIndent	Long. Gets or sets the first line indent.	Read/Write	Can be written only when formatting idle.
Font	IFontDisp. Gets or sets the standard OLE font.	Read/Write	Can be written only when formatting idle or active.
HourMinuteSeparator	String. Gets or sets the hour minute separator.	Read/Write	Can be written only when formatting idle or active.
HourType	" CRHourType ". Gets or sets the hour type.	Read/Write	Can be written only when formatting idle or active.
LeadingDayPosition	" CRLeadingDayPosition ". Gets or sets the leading day position option.	Read/Write	Can be written only when formatting idle.
LeadingDaySeparator	String. Gets or sets the leading day separator.	Read/Write	Can be written only when formatting idle or active.
LeadingDayType	" CRLeadingDayType ". Gets or sets the leading day type.	Read/Write	Can be written only when formatting idle or active.
LeftIndent	Long. Gets or sets the left indent, in twips.	Read/Write	Can be written only when formatting idle.
LineSpacing	Double. Gets the line spacing.	Read Only	None
LineSpacingType	" CRLineSpacingType ". Gets the line spacing type.	Read Only	None
MaxNumberOfLines	Integer. Gets or sets the maximum number of line for a string memo field.	Read/Write	Can be written only when formatting idle or active.
MinuteSecondSeparator	String. Gets or sets minute second separator.	Read/Write	Can be written only when formatting idle or active.
MinuteType	" CRMinuteType ". Gets or sets the minute type.	Read/Write	Can be written only when formatting idle or active.

Property	Description	Read/Write	Restriction in event handler
MonthType	" CRMonthType ". Gets or sets month type.	Read/Write	Can be written only when formatting idle or active.
NegativeType	" CRNegativeType ". Gets or sets number negative type.	Read/Write	Can be written only when formatting idle or active.
Parent	" TextObject Object ". Reference to the parent object.	Read only	Can be written only when formatting idle or active.
PmString	String. Gets or sets the PM string.	Read/Write	Can be written only when formatting idle or active.
RightIndent	Long. Gets or sets the right indent, in twips.	Read/Write	Can be written only when formatting idle.
RoundingType	" CRRoundingType ". Gets or sets the number rounding type.	Read/Write	Can be written only when formatting idle or active.
SecondType	" CRSecondType ". Gets or sets the seconds type.	Read/Write	Can be written only when formatting idle or active.
Suppress	Boolean. Gets or sets the object visibility.	Read/Write	Can be written only when formatting idle or active.
SuppressIf Duplicated	Boolean. Gets or sets the suppress if duplicate option.	Read/Write	Can be written only when formatting idle or active.
SuppressIfZero	Boolean. Gets or sets the suppress if zero option.	Read/Write	Can be written only when formatting idle or active.
TextFormat	" CRTextFormat ". Gets or sets the text format option for string memo fields.	Read/Write	Can be written only when formatting idle.
Thousands Separators	Boolean. Gets or sets the enable thousands separators option.	Read/Write	Can be written only when formatting idle or active.
ThousandSymbol	String. Gets or sets the thousand separator symbol.	Read/Write	Can be written only when formatting idle or active.
TimeBase	" CRTimeBase ". Gets or sets the time base.	Read/Write	Can be written only when formatting idle or active.
UseLeadingZero	Boolean. Gets or sets the number uses leading zero option.	Read/Write	Can be written only when formatting idle or active.
UseOneSymbol PerPage	Boolean. Gets or sets the use one symbol per page option.	Read/Write	Can be written only when formatting idle or active.
UseSystemDefaults	Boolean. Gets or sets the use system defaults formatting option.	Read/Write	Can be written only when formatting idle or active.
YearType	" CRYearType ". Gets or sets the year type.	Read/Write	Can be written only when formatting idle or active.
ZeroValueString	String. Gets or sets the zero value string for number field format.	Read/Write	Can be written only when formatting idle or active.

Remarks

After getting the field definition object from the FieldDefiniton property, you can use the Kind property to determine what type of field definition object it is. All field definition objects have the Kind property.

FieldElement Object Methods

The following method is discussed in this section:

- [“SetLineSpacing Method \(FieldElement Object\)”](#) on page 66

SetLineSpacing Method (FieldElement Object)

Use the SetLineSpacing method to get and set the line spacing and line spacing type.

Syntax

```
Sub SetLineSpacing (LineSpacing As Double,  
LineSpacingType As CRLineSpacingType)
```

Parameters

Parameter	Description
LineSpacing	Specifies the line spacing.
LineSpacingType	Specifies the line spacing type. Use one of “CRLineSpacingType” .

FieldElements Collection

The FieldElements collection contains the FieldElement objects defined in a text field. Access a specific object in the collection using the Item property.

FieldElements Collection Properties

Property	Definition	Read/Write	Restriction in event handler
Count	Long. Gets the number of objects in the collection.	Read only	None
Item (indexAs Long)	“FieldElement Object” . Gets an item from the Collection specified by the parameter index.	Read only	None
Parent	“TextObject Object” . Gets reference to the parent object.	Read only	None

FieldElements Collection Methods

The following methods are discussed in this section:

- “Add Method (FieldElements Collection)” on page 67
- “Delete Method (FieldElements Collection)” on page 67

Add Method (FieldElements Collection)

The Add method is used to add a FieldElement object to the Collection.

Syntax

Function Add (position As Long, Field)

Parameter	Description
position	Specifies the index where you would like to add the FieldElement to the Collection.
Field	Specifies the FieldElement that you want to add to the Collection.

Delete Method (FieldElements Collection)

Use the Delete method to remove a FieldElement Object from the Collection.

Syntax

Sub Delete (index)

Parameter

Parameter	Description
index	Specifies the index number of the Object that you want to remove from the Collection.

FieldMappingData Object

The FieldMappingData Object provides information related to FieldMapping Events. This Object can be accessed through the Report Object FieldMapping Event.

FieldMappingData Object Properties

Property	Description	Read/Write	Restriction in event handler
FieldName	String. Gets or sets the field name.	Read/Write	None
MappingTo FieldIndex	Integer. Gets or sets the index of the mapping to field index in the field list of the new database.	Read/Write	None

Property	Description	Read/Write	Restriction in event handler
TableName	String. Gets or sets field's table name.	Read/Write	None
ValueType	" CRFieldValueType ". Gets the value type that is in the field.	Read only	None

FieldObject Object

The FieldObject Object represents a field found in a report (for example, special field, database field, parameter field, etc.). This object provides properties for retrieving information for a field in your report. A FieldObject Object is obtained from the Item property of the "[ReportObjects Collection](#)" on [page 133](#), (for example, `ReportObjects.Item(Index)`, where the index can be the 1-based index number of the item or an Object name).

FieldObject Object Properties

Property	Description	Read/Write	Restriction in event handler
AmPmType	" CRAMPMType ". Gets or sets the AM/PM type option.	Read/Write	Can be written only when formatting idle or active.
AmString	String. Gets or sets the AM string.	Read/Write	Can be written only when formatting idle or active.
BackColor	OLE_COLOR. Gets or sets the object background color.	Read/Write	Can be written only when formatting idle or active.
BooleanOutput Type	" CRBooleanFieldFormatConditionFormulaType ". Gets or sets the Boolean output type.	Read/Write	Can be written only when formatting idle or active.
BorderColor	OLE_COLOR. Gets or sets the object border color.	Read/Write	Can be written only when formatting idle or active.
BottomLineStyle	" CRLLineStyle ". Gets or sets bottom line style.	Read/Write	Can be written only when formatting idle or active.
CanGrow	Boolean. Gets or sets can the grow option.	Read/Write	Can be written only when formatting idle or active.
CharacterSpacing	Long. Gets or sets the character spacing.	Read/Write	Can be written only when formatting idle.
CloseAtPageBreak	Boolean. Gets or sets the close border on page break option.	Read/Write	Can be written only when formatting idle or active.

Property	Description	Read/Write	Restriction in event handler
ConditionFormula	String. Gets or sets the condition formula. The condition formula can be based on recurring records or on summary fields, but it cannot be based on print-time fields, such as running totals or print-time formulas. Condition formulas cannot have shared variables.	Read/Write	Can be written only when formatting idle.
CssClass	String. Gets or sets the cascading style sheet Class.	Read/Write	Can be written only when formatting idle.
CurrencyPosition Type	" CRCurrencyPositionType ". Gets or sets the currency position type.	Read/Write	Can be written only when formatting idle or active.
CurrencySymbol	String. Gets or sets the currency symbol.	Read/Write	Can be written only when formatting idle or active.
CurrencySymbol Type	" CRCurrencySymbolType ". Gets or sets the currency symbol type.	Read/Write	Can be written only when formatting idle or active.
DateCalendarType	" CRDateCalendarType ". Gets or sets the date calendar type.	Read/Write	Can be written only when formatting idle.
DateEraType	" CRDateEraType ". Gets or sets the date era type.	Read/Write	Can be written only when formatting idle.
DateFirstSeparator	String. Gets or sets the date first separator.	Read/Write	Can be written only when formatting idle or active.
DateOrder	" CRDateOrder ". Gets or sets the date order.	Read/Write	Can be written only when formatting idle or active.
DatePrefix Separator	String. Gets or sets the date prefix separator.	Read/Write	Can be written only when formatting idle or active.
DateSecond Separator	String. Gets or sets the date second separator.	Read/Write	Can be written only when formatting idle or active.
DateSuffix Separator	String. Gets or sets the date suffix separator	Read/Write	Can be written only when formatting idle or active.
DateWindows DefaultType	" CRDateTimeFieldFormatConditionFormulaType ". Gets or sets the date windows default type.	Read/Write	Can be written only when formatting idle.
DayType	" CRDayType ". Gets or sets the day type.	Read/Write	Can be written only when formatting idle or active.
DecimalPlaces	Integer. Gets or sets the number decimal places.	Read/Write	Can be written only when formatting idle or active.
DecimalSymbol	String. Gets or sets the decimal symbol.	Read/Write	Can be written only when formatting idle or active.

Property	Description	Read/Write	Restriction in event handler
DisplayReverse Sign	Boolean. Gets or sets the reverse sign option.	Read/Write	Can be written only when formatting idle or active.
EnableTight Horizontal	Boolean. Gets or sets the tight horizontal option.	Read/Write	Can be written only when formatting idle or active.
Field	Object. Gets the field definition object (for example, ParameterFieldDefinition Object).	Read only	None
FirstLineIndent	Long. Gets or sets the first line indent.	Read/Write	Can be written only when formatting idle.
Font	IFontDisp. Gets or sets the standard OLE font.	Read/Write	Can be written only when formatting idle or active.
HasDropShadow	Boolean. Gets or sets the border drop shadow option.	Read/Write	Can be written only when formatting idle or active.
Height	Long. Gets or sets object height, in twips.	Read/Write	Can be written only when formatting idle or active.
HorAlignment	" CRAlignment ". Gets or sets the horizontal alignment.	Read/Write	Can be written only when formatting idle or active.
HourMinute Separator	String. Gets or sets the hour minute separator.	Read/Write	Can be written only when formatting idle or active.
HourType	" CRHourType ". Gets or sets the hour type.	Read/Write	Can be written only when formatting idle or active.
KeepTogether	Boolean. Gets or sets the keep object together option.	Read/Write	Can be written only when formatting idle or active.
Kind	" CROBJECTKind ". Gets which kind of object (for example, box, cross-tab, field.)	Read only	None
LeadingDay Position	" CRLeadingDayPosition ". Gets or sets the leading day position option.	Read/Write	Can be written only when formatting idle.
LeadingDay Separator	String. Gets or sets the leading day separator.	Read/Write	Can be written only when formatting idle or active.
LeadingDayType	" CRLeadingDayType ". Gets or sets the leading day type.	Read/Write	Can be written only when formatting idle or active.
Left	Long. Gets or sets the object upper left position, in twips.	Read/Write	Can be written only when formatting idle or active.
LeftIndent	Long. Gets or sets the left indent, in twips.	Read/Write	Can be written only when formatting idle.
LeftLineStyle	" CRLLineStyle ". Gets or sets the left line style.	Read/Write	Can be written only when formatting idle or active.
LineSpacing	Double. Gets the line spacing.	Read Only	None

Property	Description	Read/Write	Restriction in event handler
LineSpacingType	" CRLineSpacingType ". Gets the line spacing type.	Read Only	None
MaxNumberOfLines	Integer. Gets or sets the maximum number of line for a string memo field.	Read/Write	Can be written only when formatting idle or active.
MinuteSecondSeparator	String. Gets or sets minute second separator.	Read/Write	Can be written only when formatting idle or active.
MinuteType	" CRMinuteType ". Gets or sets the minute type.	Read/Write	Can be written only when formatting idle or active.
MonthType	" CRMonthType ". Gets or sets month type.	Read/Write	Can be written only when formatting idle or active.
Name	String. Gets or sets the object name.	Read/Write	Can be written only when formatting idle or active.
NegativeType	" CRNegativeType ". Gets or sets number negative type.	Read/Write	Can be written only when formatting idle or active.
NextValue	Variant. Gets the field next value.	Read only	Can be written only when formatting idle or active.
Parent	" Section Object ". Reference to the parent object.	Read only	Can be written only when formatting idle or active.
PmString	String. Gets or sets the PM string.	Read/Write	Can be written only when formatting idle or active.
PreviousValue	Variant. Gets the field previous value.	Read only	Can be written only when formatting idle or active.
RightIndent	Long. Gets or sets the right indent, in twips.	Read/Write	Can be written only when formatting idle.
RightLineStyle	" CRLineStyle ". Gets or sets the right line style.	Read/Write	Can be written only when formatting idle or active.
RoundingType	" CRRoundingType ". Gets or sets the number rounding type.	Read/Write	Can be written only when formatting idle or active.
SecondType	" CRSecondType ". Gets or sets the seconds type.	Read/Write	Can be written only when formatting idle or active.
Suppress	Boolean. Gets or sets the object visibility.	Read/Write	Can be written only when formatting idle or active.
SuppressIfDuplicated	Boolean. Gets or sets the suppress if duplicate option.	Read/Write	Can be written only when formatting idle or active.
SuppressIfZero	Boolean. Gets or sets the suppress if zero option.	Read/Write	Can be written only when formatting idle or active.
TextColor	OLE_COLOR. Gets or sets the object text color.	Read/Write	Can be written only when formatting idle or active.

Property	Description	Read/Write	Restriction in event handler
TextFormat	" CRTextFormat ". Gets or sets the text format option for string memo fields.	Read/Write	Can be written only when formatting idle.
TextRotationAngle	" CRRotationAngle ". Gets or sets the text rotation angle.	Read/Write	Can be written only when formatting idle.
Thousands Separators	Boolean. Gets or sets the enable thousands separators option.	Read/Write	Can be written only when formatting idle or active.
ThousandSymbol	String. Gets or sets the thousand separator symbol.	Read/Write	Can be written only when formatting idle or active.
TimeBase	" CRTTimeBase ". Gets or sets the time base.	Read/Write	Can be written only when formatting idle or active.
Top	Long. Gets or sets the object upper top position, in twips.	Read/Write	Can be written only when formatting idle or active.
TopLineStyle	" CRLLineStyle ". Gets or sets the top line style.	Read/Write	Can be written only when formatting idle or active.
UseLeadingZero	Boolean. Gets or sets the number uses leading zero option.	Read/Write	Can be written only when formatting idle or active.
UseOneSymbolPer Page	Boolean. Gets or sets the use one symbol per page option.	Read/Write	Can be written only when formatting idle or active.
UseSystemDefaults	Boolean. Gets or sets the use system defaults formatting option.	Read/Write	Can be written only when formatting idle or active.
Value	Variant. Gets the field current value.	Read Only	Can be written only when formatting idle or active.
Width	Long. Gets or sets the object width, in twips.	Read/Write	Can be written only when formatting idle or active.
YearType	" CRYearType ". Gets or sets the year type.	Read/Write	Can be written only when formatting idle or active.
ZeroValueString	String. Gets or sets the zero value string for number field format.	Read/Write	Can be written only when formatting idle or active.

FieldObject Object Methods

The following methods are discussed in this section:

- "[SetLineSpacing Method \(FieldObject Object\)](#)" on page 72
- "[SetUnboundFieldSource Method \(FieldObject Object\)](#)" on page 73

SetLineSpacing Method (FieldObject Object)

Use the SetLineSpacing method to get and set the line spacing and line spacing type.

Syntax

```
Sub SetLineSpacing (LineSpacing As Double,  
LineSpacingType As CRLineSpacingType)
```

Parameters

Parameter	Description
LineSpacing	Specifies the line spacing.
LineSpacingType	Specifies the line spacing type. Use one of "CRLineSpacingType".

SetUnboundFieldSource Method (FieldObject Object)

Use the SetUnbound FieldSource Method to bind a DataSource to an unbound field.

Syntax

```
Sub SetUnboundFieldSource (pUnboundFieldSource As String)
```

Parameter

Parameter	Description
pUnboundFieldSource	BSTR specifies the datasouce, Crystal formula format. See Remarks below.

Remarks

For example:

```
object.SetUnboundFieldSource("{Customer.CustomerID}")
```

FormattingInfo Object

The FormattingInfo object contains information about the section currently being formatted.

FormattingInfo Object Properties

Property	Description	Read/Write	Restriction in event handler
IsEndOfGroup	Boolean. Gets whether the current formatting section is the end of a group.	Read only	None
IsRepeatedGroup Header	Boolean. Gets whether the current formatting section is a repeated group header.	Read only	None
IsStartOfGroup	Boolean. Gets whether the current formatting section is the start of a group.	Read only	None

FormulaFieldDefinition Object

The FormulaFieldDefinition Object provides properties and methods for retrieving information and setting options for any named formula field in a report.

FormulaFieldDefinition Object Properties

Property	Description	Read/Write	Restriction in event handler
FormulaField Name	String. Gets the formula field name as it appears in the RDC Dataview Panel.	Read only	None
Kind	" CRFieldKind ". Gets what kind of field (for example, database, summary, formula, etc.).	Read only	None
Name	String. Gets the unique name of the formula field in Crystal formula format as it would be referenced in the report (for example, {@ExampleFormula}).	Read only	None
NextValue	Variant. Gets the field next value.	Read only	Can be read only when top-level Report object is formatting active.
NumberOf Bytes	Integer. Gets the number of bytes required to store the field data in memory.	Read only	None
Parent	" Report Object ". Gets reference to the parent object.	Read only	None
PreviousValue	Variant. Gets the field previous value.	Read only	Can be read only when top-level Report object is formatting active.
Text	String. Gets or sets the text of the formula. The formula text is changed immediately in the report. If you generate a report with an invalid formula, you may receive an exception error. Syntax can be Crystal Refort or Visual Basic. See Remarks below.	Read/Write	Can be written only when formatting idle.
Value	Variant. Gets the field current value.	Read only	Can be read only when top-level Report object is formatting active.
ValueType	" CRFieldValueType ". Gets which type of value is found in the field.	Read only	None

Remarks

Crystal Reports 9 supports formulas in Crystal Reports syntax and Visual Basic syntax. When setting text to a formula, the syntax is determined by the FormulaSyntax property of the parent "**Report Object**". The default syntax is Crystal Report syntax (crCrystalSyntaxFormula).

FormulaFieldDefinition Object Methods

The following method is discussed in this section:

- “[Check Method \(FormulaFieldDefinition Object\)](#)” on page 75

Check Method (FormulaFieldDefinition Object)

The Check method checks formulas for errors (syntax errors).

Syntax

```
Sub Check (pBool As Boolean, ppErrorString As String)
```

Parameters

Parameter	Description
pBool	Boolean value indicating the condition of the formula string. Will be set to TRUE if the formula is valid and FALSE if the formula contains one or more errors.
ppErrorString	Specifies the error message string if the formula contains an error.

FormulaFieldDefinitions Collection

The FormulaFieldDefinitions Collection is a collection of named formulas in the report. Access a specific “[FormulaFieldDefinition Object](#)”, in the collection using the Item property.

FormulaFieldDefinitions Collection Properties

Property	Description	Read/Write	Restriction in event handler
Count	Long. Gets the number of formula field definitions in the collection.	Read only	None
Item (index As Long)	“ FormulaFieldDefinitions Collection ”. Gets collection item. See Remarks below.	Read only	None
Parent	“ Report Object ”. Gets reference to the parent object.	Read only	None

Remarks

Instead of using the Item property as shown, you can reference a formula field directly (for example, FormulaFieldDefinitions(1)).

FormulaFieldDefinitions Collection Methods

The following methods are discussed in this section:

- “[Add Method \(FormulaFieldDefinitions Collection\)](#)” on page 76

- “Delete Method (FormulaFieldDefinitions Collection)” on page 76
- “GetItemByName Method (FormulaFieldDefinitions Collection)” on page 76

Add Method (FormulaFieldDefinitions Collection)

Use the Add method to add the specified formula field to the FormulaFieldDefinitions Collection.

Syntax

```
Function Add (formulaName As String,  
            Text As String) As FormulaFieldDefinition
```

Parameters

Parameter	Description
formulaName	Specifies the formula field that you want to add to the Collection.
Text	Specifies the text of the formula field that you want to add.

Returns

Returns a FormulaFieldDefinition member of the Collection.

Delete Method (FormulaFieldDefinitions Collection)

Use the Delete method to remove the specified formula field from the FormulaFieldDefinitions Collection.

Syntax

```
Sub Delete (index)
```

Parameter

Parameter	Description
index	Specifies the formula field that you want to delete from the Collection.

GetItemByName Method (FormulaFieldDefinitions Collection)

Use the GetItemByName method to retrieve a FormulaFieldDefinition object by name.

Syntax

```
Sub GetItemByName(name As String, formulaFieldDefinition)
```

Parameters

Parameter	Description
name	The item's unique name.
formulaFieldDefinition	A FormulaFieldDefinition object to hold the retrieved item.

GraphObject Object

The GraphObject Object represents a graph/chart found in a report. This object provides properties for retrieving information and setting options for a graph in your report (that is, graph data type — group, detail or graph display type — bar, pie, and so on.).

GraphObject Object Properties

Examples

“How to format a graph object (chart)” on page 185

Property	Description	Read/Write	Restriction in event handler
AutoRangeData2Axis	Boolean. Gets or sets the auto range option for data2 axis. See Remarks below.	Read/Write	Can be written only when formatting idle.
AutoRangeDataAxis	Boolean. Gets or sets the auto range option for data axis. See Remarks below.	Read/Write	Can be written only when formatting idle.
AutoRangeSeriesAxis	Boolean. Gets or sets the auto range option for series axis.	Read/Write	Can be written only when formatting idle.
BackColor	OLE_COLOR. Gets or sets the object background color.	Read/Write	Can be written only when formatting idle or active.
BarSize	“CRBarSize”. Gets or sets the bar size.	Read/Write	Can be written only when formatting idle.
BorderColor	OLE_COLOR. Gets or sets the object border color.	Read/Write	Can be written only when formatting idle or active.
BottomLineStyle	“CRLLineStyle”. Gets or sets the bottom line style.	Read/Write	Can be written only when formatting idle or active.
CloseAtPageBreak	Boolean. Gets or sets the close border on page break option.	Read/Write	Can be written only when formatting idle or active.
ConditionFields	“FieldDefinitionsCollection”. Gets the condition fields Collection.	Read only	None
ConditionFormula	String. Gets or sets the condition formula. The condition formula can be based on recurring records or on summary fields, but it cannot be based on print-time fields, such as running totals or print-time formulas. Condition formulas cannot have shared variables.	Read/Write	Can be written only when formatting idle.

Property	Description	Read/Write	Restriction in event handler
CrossTabObject	"CrossTabObject Object". Gets the crosstab object if this is a CrossTab chart.	Read only	None
CssClass	String. Gets or sets the cascading style sheet Class.	Read/Write	Can be written only when formatting idle.
Data2Axis DivisionMethod	"CRDivisionMethod". Gets or sets the data2 axis division method.	Read/Write	Can be written only when formatting idle.
Data2Axis DivisionNumber	Long. Gets or sets the data2 axis division number.	Read/Write	Can be written only when formatting idle.
Data2Axis Gridline	"CRGridlineType". Gets or sets the data2 axis grid line type.	Read/Write	Can be written only when formatting idle.
Data2Axis NumberFormat	"CRNumberFormat". Gets or sets the data2 axis number format.	Read/Write	Can be written only when formatting idle.
DataAxis DivisionMethod	"CRDivisionMethod". Gets or sets the data axis division method.	Read/Write	Can be written only when formatting idle.
DataAxis DivisionNumber	Long. Gets or sets the data axis division number.	Read/Write	Can be written only when formatting idle.
DataAxis Gridline	"CRGridlineType". Gets or sets the data axis grid line type.	Read/Write	Can be written only when formatting idle.
DataAxis NumberFormat	"CRNumberFormat". Gets or sets the data axis number format.	Read/Write	Can be written only when formatting idle.
DataLabelFont	IFontDisp. Gets or sets standard OLE font for chart data labels.	Read/Write	Can be written only when formatting idle.
DataPoint	"CRGraphDataPoint". Gets or sets the graph data points on risers.	Read/Write	Can be written only when formatting idle.
DataTitleFont	IFontDisp. Gets or sets standard OLE font for chart data title.	Read/Write	Can be written only when formatting idle.
DataType	"CRGraphDataType". Returns which type of data used in the graph.	Read only	None
DataValue NumberFormat	"CRNumberFormat". Gets or sets the data value number format.	Read/Write	Can be written only when formatting idle.
EnableAuto ScaleDataAxis	Boolean. Gets or sets the data axis auto-scale option.	Read/Write	Can be written only when formatting idle.
EnableAuto ScaleData2Axis	Boolean. Gets or sets the data2 axis auto-scale option.	Read/Write	Can be written only when formatting idle.
EnableAutoScale SeriesAxis	Boolean. Gets or sets the series axis auto-scale option.	Read/Write	Can be written only when formatting idle.
EnableFor EachRecord	Boolean. Gets or sets the enable for each record option.	Read/Write	Can be written only when formatting idle.

Property	Description	Read/Write	Restriction in event handler
EnableShowLegend	Boolean. Gets or sets the show legend option.	Read/Write	Can be written only when formatting idle.
EnableSummarizeValues	Boolean. Gets or sets the enable summarize values option.	Read/Write	Can be written only when formatting idle.
FootNote	String. Gets or sets the footnote.	Read/Write	Can be written only when formatting idle.
FootnoteFont	IFontDisp. Gets or sets standard OLE font for chart footnote.	Read/Write	Can be written only when formatting idle.
GraphColor	crgraphcolor. Gets or sets the graph color.	Read/Write	Can be written only when formatting idle.
GraphDirection	"CRGraphDirection". Gets or sets the graph direction.	Read/Write	Can be written only when formatting idle.
GraphType	"CRGraphType". Gets or sets the graph type.	Read/Write	Can be written only when formatting idle.
GroupAxisGridline	"CRGridlineType". Gets or sets the group axis grid line type.	Read/Write	Can be written only when formatting idle.
GroupLabelFont	IFontDisp. Gets or sets standard OLE font for chart group labels.	Read/Write	Can be written only when formatting idle.
GroupsTitle	String. Gets or sets the groups title.	Read/Write	Can be written only when formatting idle.
GroupTitleFont	IFontDisp. Gets or sets standard OLE font for chart group title.	Read/Write	Can be written only when formatting idle.
HasDropShadow	Boolean. Gets or sets the border drop shadow option.	Read/Write	Can be written only when formatting idle or active.
Height	Long. Gets or sets the object height, in twips.	Read/Write	Can be written only when formatting idle or active.
IsFootnoteByDefault	Boolean. Gets or sets footnote default flag.	Read/Write	Can be written only when formatting idle or active.
IsGroupsTitleByDefault	Boolean. Gets or set groups title default flag.	Read/Write	Can be written only when formatting idle or active.
IsSeriesTitleByDefault	Boolean. Gets or sets series title default flag.	Read/Write	Can be written only when formatting idle or active.
IsSubTitleByDefault	Boolean. Get or sets subtitle default flag.	Read/Write	Can be written only when formatting idle or active.

Property	Description	Read/Write	Restriction in event handler
IsTitleByDefault	Boolean. Gets or sets title default flag.	Read/Write	Can be written only when formatting idle or active.
IsXAxisTitleByDefault	Boolean. Gets or sets X axis title default flag.	Read/Write	Can be written only when formatting idle or active.
IsYAxisTitleByDefault	Boolean. Gets or sets Y axis title default flag.	Read/Write	Can be written only when formatting idle or active.
IsZAxisTitleByDefault	Boolean. Gets or sets Z axis title default flag.	Read/Write	Can be written only when formatting idle or active.
KeepTogether	Boolean. Gets or sets keep object together option.	Read/Write	Can be written only when formatting idle or active.
Kind	" CRObjektKind ". Gets which kind of object (for example, box, cross-tab).	Read only	None
Left	Long. Gets or sets the object upper left position, in twips.	Read/Write	Can be written only when formatting idle or active.
LeftLineStyle	" CRLineStyle ". Gets or sets the left line style.	Read/Write	Can be written only when formatting idle or active.
LegendFont	IFontDisp. Gets or sets standard OLE font for legend.	Read/Write	Can be written only when formatting idle.
LegendLayout	" CRPieLegendLayout ". Gets or sets the legend layout for a pie chart.	Read/Write	Can be written only when formatting idle.
LegendPosition	" CRLegendPosition ". Gets or sets the legend position.	Read/Write	Can be written only when formatting idle.
MarkerShape	" CRMarkerShape ". Gets or sets the marker shape.	Read/Write	Can be written only when formatting idle.
MarkerSize	" CRMarkerSize ". Gets or sets the marker size.	Read/Write	Can be written only when formatting idle.
MaxData2Axis Value	Double. Gets or sets data2-axis max value. See Remarks below.	Read/Write	Can be written only when formatting idle.
MaxDataAxis Value	Double. Gets or sets data-axis max value. See Remarks below.	Read/Write	Can be written only when formatting idle.
MaxSeriesAxis Value	Double. Gets or sets series axis max value. See Remarks below.	Read/Write	Can be written only when formatting idle.

Property	Description	Read/Write	Restriction in event handler
MinData2Axis Value	Double. Gets or sets data2-axis min value. See Remarks below.	Read/Write	Can be written only when formatting idle.
MinDataAxis Value	Double. Gets or sets data-axis min value. See Remarks below.	Read/Write	Can be written only when formatting idle.
MinSeriesAxis Value	Double. Gets or sets series axis min value. See Remarks below.	Read/Write	Can be written only when formatting idle.
Name	String. Gets or sets the object name.	Read/Write	Can be written only when formatting idle.
Parent	"Section Object". Gets reference to the parent object.	Read only	None
PieSize	"CRPieSize". Gets or sets the size for pie charts.	Read/Write	Can be written only when formatting idle.
RightLineStyle	"CRLLineStyle". Gets or sets the right line style.	Read/Write	Can be written only when formatting idle or active.
SeriesAxis DivisionMethod	"CRDivisionMethod". Gets or sets the series axis division method.	Read/Write	Can be written only when formatting idle.
SeriesAxis DivisionNumber	Long. Gets or sets the series axis division number.	Read/Write	Can be written only when formatting idle.
SeriesAxis NumberFormat	"CRNumberFormat". Gets or sets the series axis number format.	Read/Write	Can be written only when formatting idle.
SeriesAxis Gridline	"CRGridlineType". Gets or sets the series axis grid line type.	Read/Write	Can be written only when formatting idle.
SeriesLabelFont	IFontDisp. Gets or sets standard OLE font for chart series labels.	Read/Write	Can be written only when formatting idle.
SeriesTitle	String. Gets or sets the series title.	Read/Write	Can be written only when formatting idle.
SeriesTitleFont	IFontDisp. Gets or sets standard OLE font for chart series title.	Read/Write	Can be written only when formatting idle.
SliceDetachment	"CRSliceDetachment". Gets or sets the slice detachment.	Read/Write	Can be written only when formatting idle.
SubTitle	String. Gets or sets subtitle.	Read/Write	Can be written only when formatting idle.
SubTitleFont	IFontDisp. Gets or sets standard OLE font for chart subtitle.	Read/Write	Can be written only when formatting idle.
SummaryFields	"SummaryFieldDefinitions Collection". Gets the summary fields Collection.	Read only	None
Suppress	Boolean. Gets or sets the object visibility.	Read/Write	Can be written only when formatting idle or active.

Property	Description	Read/Write	Restriction in event handler
Title	String. Gets or sets the title.	Read/Write	Can be written only when formatting idle.
TitleFont	IFontDisp. Gets or sets standard OLE font for chart title.	Read/Write	Can be written only when formatting idle.
Top	Long. Gets or sets the object upper top position, in twips.	Read/Write	Can be written only when formatting idle or active.
TopLineStyle	" CRLLineStyle ". Gets or sets the top line style.	Read/Write	Can be written only when formatting idle or active.
ViewingAngle	" CRViewingAngle ". Gets or sets the viewing angle.	Read/Write	Can be written only when formatting idle.
Width	Long. Gets or sets the object width, in twips.	Read/Write	Can be written only when formatting idle or active.
XAxisTitle	String. Gets or sets the title for X axis.	Read/Write	Can be written only when formatting idle.
Y2AxisTitle	String. Gets or sets the title Y2 axis.	Read/Write	Can be written only when formatting idle.
YAxisTitle	String. Gets or sets the title for Y axis.	Read/Write	Can be written only when formatting idle.
ZAxisTitle	String. Gets or sets the title for Z axis.	Read/Write	Can be written only when formatting idle.

Remarks

Properties Max/MinData/SeriesAxisValue will be ignored if the corresponding AutoRangeData/SeriesAxis property is set to TRUE. If the Max/MinDataDataAxis/Series properties are set at runtime, then the corresponding AutoRangeData/SeriesAxis must be set to FALSE.

This property must be set to FALSE...	...or this property will be ignored
AutoRangeData2Axis	MaxData2AxisValue
AutoRangeDataAxis	MaxDataAxisValue
AutoRangeData2Axis	MinData2AxisValue
AutoRangeDataAxis	MinDataAxisValue
AutoRangeSeriesAxis	MaxSeriesAxisValue
AutoRangeSeriesAxis	MinSeriesAxisValue

GroupNameFieldDefinition Object

The GroupNameFieldDefinition Object provides properties and methods for retrieving information on a group name field found in a report (for example, number of group, value type). A GroupNameFieldDefinition Object can be obtained from the Field property of the “FieldMappingData Object” when the specified field is a group name field or from a “GroupNameFieldDefinitions Collection” retrieved from the GroupNameFields property of the “Report Object”.

GroupNameFieldDefinition Object Properties

Examples

“How to format a group in the section format event” on page 200

Property	Description	Read/Write	Restriction in event handler
GroupNameConditionFormula	String. Gets or sets the group name condition formula.	Read/Write	Can be written only when formatting idle.
GroupNameFieldName	String. Gets the group name field name.	Read only	None
GroupNumber	Integer. If the area is a group, this gets the group number. Otherwise, exception is thrown.	Read only	None
Kind	“CRFieldKind”. Gets which kind of field (for example, database, summary, formula).	Read only	None
Name	String. Gets the field definition unique formula name in Crystal Report formula syntax.	Read only	None
NextValue	Variant. Gets the field next value.	Read only	Can be read only when top-level Report object is formatting active.
NumberOfBytes	Integer. Gets the number of bytes required to store the field data in memory.	Read only	None
Parent	“Report Object”. Gets reference to the parent object.	Read only	None
PreviousValue	Variant. Gets the field previous value.	Read only	Can be read only when top-level Report object is formatting active.
Value	Variant. Gets the field current value.	Read only	Can be read only when top-level Report object is formatting active.
ValueType	“CRFieldValueType”. Gets which type of value is found in the field.	Read only	None

GroupNameFieldDefinitions Collection

The GroupNameFieldDefinitions Collection is a collection of named groups in the report. Access a specific “GroupNameFieldDefinition Object” in the collection using the Item property.

GroupNameFieldDefinitions Collection Properties

Property	Description	Read/Write	Restriction in event handler
Count	Long. Gets the number of group name field definitions in the Collection.	Read only	None
Item (indexAs Long)	“GroupNameFieldDefinition Object”. Gets the specified object from the Collection. Item has an index parameter that is the 1-based index number of the Object in the Collection. The items in the collection are indexed in the order they were added to the report.	Read only	None
Parent	“Report Object”. Gets reference to the parent object.	Read only	None

Remarks

Instead of using the Item property as shown, you can reference a group name field directly (for example, GroupNameFieldDefinitions(1)).

LineObject Object

The LineObject Object represents a line drawn on a report. This object provides properties for getting information for lines on a report.

LineObject Object Properties

Property	Description	Read/Write	Restriction in event handler
Bottom	Long. Gets or sets the line lower bottom position, in twips.	Read/Write	Can be written only when formatting idle.
Condition Formula	String. Gets or sets the condition formula. The condition formula can be based on recurring records or on summary fields, but it cannot be based on print-time fields, such as running totals or print-time formulas. Condition formulas cannot have shared variables.	Read/Write	Can be written only when formatting idle.

Property	Description	Read/Write	Restriction in event handler
CssClass	String. Gets or sets the cascading style sheet Class.	Read/Write	Can be written only when formatting idle.
EndSection	"Section Object". Gets the end section.	Read only	None
ExtendToBottomOfSection	Boolean. Gets or sets the extend to bottom of section option.	Read/Write	Can be written only when formatting idle.
Kind	"CRObjectKind". Gets which kind of object (for example, box, cross-tab, field).	Read only	None
Left	Long. Gets or sets the object upper left position, in twips.	Read/Write	Can be written only when formatting idle.
LineColor	OLE_COLOR. Gets or sets the line color.	Read/Write	Can be written only when formatting idle.
LineStyle	"CRLineStyle". Gets or sets the line style. See Remarks below.	Read/Write	Can be written only when formatting idle.
LineThickness	Long. Gets or sets the line thickness, in twips.	Read/Write	Can be written only when formatting idle.
Name	String. Gets or sets object name.	Read/Write	Can be written only when formatting idle.
Parent	"Section Object". Gets reference to the parent object.	Read only	None
Right	Long. Gets or sets the line lower right position, in twips.	Read/Write	Can be written only when formatting idle.
Suppress	Boolean. Gets or sets the object visibility.	Read/Write	Can be written only when formatting idle.
Top	Long. Gets or sets the object upper top position, in twips.	Read/Write	Can be written only when formatting idle.

Remarks

For property LineStyle, crLSDoubleLine and crLSNoLine are not valid.

MapObject Object

The MapObject Object represents a geographic map object in a report. This object provides properties for getting information for Map objects in a report.

MapObject Object Properties

Examples

“How to format a map object” on page 187

Property	Description	Read/Write	Restriction in event handler
BackColor	OLE_COLOR. Gets or sets the object background color.	Read/Write	Can be written only when formatting idle or active.
BorderColor	OLE_COLOR. Gets or sets the object border color.	Read/Write	Can be written only when formatting idle or active.
BottomLineStyle	“CRLinestyle”. Gets or sets the bottom line style.	Read/Write	Can be written only when formatting idle or active.
CloseAtPage Break	Boolean. Gets or sets the close border on page break option.	Read/Write	Can be written only when formatting idle or active.
Condition Formula	String. Gets or sets the condition formula. The condition formula can be based on recurring records or on summary fields, but it cannot be based on print-time fields, such as running totals or print-time formulas. Condition formulas cannot have shared variables.	Read/Write	Can be written only when formatting idle.
CssClass	String. Gets or sets the cascading style sheet Class.	Read/Write	Can be written only when formatting idle.
HasDrop Shadow	Boolean. Gets or sets the border drop shadow option.	Read/Write	Can be written only when formatting idle or active.
Height	Long. Gets or sets the object height, in twips.	Read/Write	Can be written only when formatting idle or active.
KeepTogether	Boolean. Gets or sets the keep object together option.	Read/Write	Can be written only when formatting idle or active.
Kind	“CROBJECTKind”. Gets which kind of object (for example, box, cross-tab, field).	Read only	None
Left	Long. Gets or sets the object upper left position, in twips.	Read/Write	Can be written only when formatting idle or active.
LeftLineStyle	“CRLinestyle”. Gets or sets the left line style.	Read/Write	Can be written only when formatting idle or active.

Property	Description	Read/Write	Restriction in event handler
Name	String. Gets or sets the object name.	Read/Write	Can be written only when formatting idle.
Parent	"Section Object". Gets reference to the parent object.	Read only	None
RightLineStyle	"CRLLineStyle". Gets or sets the right line style.	Read/Write	Can be written only when formatting idle or active.
Suppress	Boolean. Gets or sets the object visibility.	Read/Write	Can be written only when formatting idle or active.
Top	Long. Gets or sets the object upper top position, in twips.	Read/Write	Can be written only when formatting idle or active.
TopLineStyle	"CRLLineStyle". Gets or sets the top line style.	Read/Write	Can be written only when formatting idle or active.
Width	Long. Gets or sets the object width, in twips.	Read/Write	Can be written only when formatting idle or active.

ObjectSummaryFieldDefinitions Collection

ObjectSummaryFieldDefinitions is a Collection of "SummaryFieldDefinition Object" Objects.

ObjectSummaryFieldDefinitions Collection Properties

Property	Description	Read/Write	Restriction in event handler
Count	Long. Gets the number of object summary field definitions in the Collection.	Read only	None
Item (indexAs Long)	"SummaryFieldDefinition Object". Gets the specified object from the Collection. Item has an index parameter that is the 1-based index number of the Object in the Collection. The items in the collection are indexed in the order they were added to the report.	Read only	None
Parent	"Report Object". Gets reference to the parent object.	Read only	None

ObjectSummaryFieldDefinitions Collection Methods

The following methods are discussed in this section:

- "Add Method (ObjectSummaryFieldDefinitions Collection)" on page 88
- "Delete Method (ObjectSummaryFieldDefinitions Collection)" on page 88

Add Method (ObjectSummaryFieldDefinitions Collection)

Use the Add method to add the specified object summary field to the ObjectSummaryFieldDefinitions Collection.

Syntax

```
Sub Add (summaryField)
```

Parameter

Parameter	Description
summaryField	Specifies the object summary field that you want to add to the Collection.

Delete Method (ObjectSummaryFieldDefinitions Collection)

Use the Delete method to remove the specified object summary field from the ObjectSummaryFieldDefinitions Collection.

Syntax

```
Sub Delete (index As Long)
```

Parameter

Parameter	Description
index	Specifies the object summary field that you want to delete from the Collection.

OlapGridObject Object

The OlapGridObject Object represents an Olap Object in a report.

OlapGridObject Object Properties

Examples

“How to format an olap grid object” on page 188

Property	Description	Read/Write	Restriction in event handler
BackColor	OLE_COLOR. Gets or sets the object background color.	Read/Write	Can be written only when formatting idle or active.
BorderColor	OLE_COLOR. Gets or sets the object border color.	Read/Write	Can be written only when formatting idle or active.

Property	Description	Read/Write	Restriction in event handler
BottomLineStyle	" CRLLineStyle ". Gets or sets the bottom line style.	Read/Write	Can be written only when formatting idle or active.
CloseAtPageBreak	Boolean. Gets or sets the close border on page break option.	Read/Write	Can be written only when formatting idle or active.
ConditionFormula	String. Gets or sets the condition formula. The condition formula can be based on recurring records or on summary fields, but it cannot be based on print-time fields, such as running totals or print-time formulas. Condition formulas cannot have shared variables.	Read/Write	Can be written only when formatting idle.
HasDropShadow	Boolean. Gets or sets the border drop shadow option.	Read/Write	Can be written only when formatting idle or active.
Height	Long. Gets the object height, in twips.	Read only	Can be written only when formatting idle or active.
KeepTogether	Boolean. Gets or sets the keep object together option.	Read/Write	Can be written only when formatting idle or active.
Kind	" CRObjektKind ". Gets report object kind.	Read only	None
Left	Long. Gets or sets the object upper left position, in twips.	Read/Write	Can be written only when formatting idle or active.
LeftLineStyle	" CRLLineStyle ". Gets or sets the left line style.	Read/Write	Can be written only when formatting idle or active.
Name	String. Gets or sets the object name.	Read/Write	Can be written only when formatting idle or active.
Parent	" Section Object ". Gets reference to the parent object.	Read only	None
RightLineStyle	" CRLLineStyle ". Gets or sets the right line style.	Read/Write	Can be written only when formatting idle or active.
Suppress	Boolean. Gets or sets the object visibility.	Read/Write	Can be written only when formatting idle or active.
Top	Long. Gets or sets the object upper top position, in twips.	Read/Write	Can be written only when formatting idle or active.
TopLineStyle	" CRLLineStyle ". Gets or sets the top line style.	Read/Write	Can be written only when formatting idle or active.
Width	Long. Gets the object width, in twips.	Read only	Can be written only when formatting idle or active.

OleObject Object

The OleObject Object represents an OLE object in a report. This object provides properties for getting information for OLE objects in a report.

OleObject Object Properties

Examples

“How to format an OLE object” on page 189

Property	Description	Read/Write	Restriction in event handler
BackColor	OLE_COLOR. Gets or sets the object background color.	Read/Write	Can be written only when formatting idle or active.
BorderColor	OLE_COLOR. Gets or sets the object border color.	Read/Write	Can be written only when formatting idle or active.
Bottom Cropping	Long. Gets or sets the bottom cropping size, in twips.	Read/Write	Can be written only when formatting idle.
BottomLine Style	“CRLineStyle”. Gets or sets the bottom line style.	Read/Write	Can be written only when formatting idle or active.
CloseAtPage Break	Boolean. Gets or sets the close border on page break option.	Read/Write	Can be written only when formatting idle or active.
Condition Formula	String. Gets or sets the condition formula. The condition formula can be based on recurring records or on summary fields, but it cannot be based on print-time fields, such as running totals or print-time formulas. Condition formulas cannot have shared variables.	Read/Write	Can be written only when formatting idle.
CssClass	String. Gets or sets the cascading style sheet Class.	Read/Write	Can be written only when formatting idle.
Formatted Picture	IPictureDisp. Gets or sets the specified picture during formatting.	Read/Write	Can be read or written only when top-level Report object is formatting active.
HasDrop Shadow	Boolean. Gets or sets the border drop shadow option.	Read/Write	Can be written only when formatting idle or active.
Height	Long. Gets or sets the object height, in twips.	Read/Write	Can be written only when formatting idle or active.
KeepTogether	Boolean. Gets or sets the keep object together option.	Read/Write	Can be written only when formatting idle or active.
Kind	“CRObjektKind”. Gets which kind of object (for example, box, cross-tab, field).	Read only	None

Property	Description	Read/Write	Restriction in event handler
Left	Long. Gets or sets the object upper left position, in twips.	Read/Write	Can be written only when formatting idle or active.
LeftCropping	Long. Gets or sets the left cropping size, in twips.	Read/Write	Can be written only when formatting idle.
LeftLineStyle	" CRLLineStyle ". Gets or sets the left line style.	Read/Write	Can be written only when formatting idle or active.
Name	String. Gets or sets the object name.	Read/Write	Can be written only when formatting idle.
Parent	" Section Object ". Gets reference to the parent object.	Read only	None
RightCropping	Long. Gets or sets the right cropping size, in twips.	Read/Write	Can be written only when formatting idle.
RightLineStyle	" CRLLineStyle ". Gets or sets the right line style.	Read/Write	Can be written only when formatting idle or active.
Suppress	Boolean. Gets or sets the object visibility.	Read/Write	Can be written only when formatting idle or active.
Top	Long. Gets or sets the object upper top position, in twips.	Read/Write	Can be written only when formatting idle or active.
TopCropping	Long. Gets or sets the top cropping size, in twips.	Read/Write	Can be written only when formatting idle.
TopLineStyle	" CRLLineStyle ". Gets or sets the top line style.	Read/Write	Can be written only when formatting idle or active.
Width	Long. Gets or sets the object width, in twips.	Read/Write	Can be written only when formatting idle or active.
XScaling	Double. Gets or sets the width scaling factor. For example, 1 means 100%, 2 means 200%, 0.5 means 50%. The scaling factor may range from 0.01 to 100.	Read/Write	Can be written only when formatting idle.
YScaling	Double. Gets or sets height scaling factor. For example, 1 means 100%, 2 means 200%, 0.5 means 50% etc. The scaling factor may range from 0.01 to 100.	Read/Write	Can be written only when formatting idle.

OleObject Object Methods

The following method is discussed in this section:

- "**SetOleLocation Method (OleObject Object)**" on page 92
- "**GetLinkSource Method (OleObject Object)**" on page 92

SetOleLocation Method (OleObject Object)

Use the SetOleLocation method to specify the location of an OLE Object.

Syntax

```
Sub SetOleLocation (pLocation As String)
```

Parameter

Parameter	Description
pLocation	Specifies the location of the OLE Object.

Remarks

SetOleLocation must be called from the Format Event of the Section object, and only when formatting active. For more more information see [“Format Event \(Section Object\)” on page 148](#).

GetLinkSource Method (OleObject Object)

Use the GetLinkSource method to retrieve the linked OLE object path and file name.

Syntax

```
Function GetLinkSource() As String
```

Page Object

The Page Object is part of the Page Engine. Use the Page Engine when designing web sites using Active Server Pages, the Crystal Report Engine Automation Server, and the Crystal Design-Time ActiveX Control. Unless you are experienced with the Crystal Report Engine Object Library, you should allow the Crystal Design-Time ActiveX Control to generate VBScript code in your Active Server Pages that controls the Page Engine objects.

The Page Engine generates pages of a report on the web server and sends the pages to client web browsers as they are requested. For example, when a user first requests a report, only the first page is sent to the web browser. If the user pages forward or backward in the report, or requests a specific page, only that page is sent. This limits the resources required by the web server and reduces download time for the client browser. A Page Object is a single generated page that is sent to the browser.

Page Object Properties

Property	Description	Read/Write	Restriction in event handler
IsLastPage	Boolean. Gets whether the page generated is the last page of the report.	Read only	None
IsMissingTotalPage Count	Boolean. Gets whether the page misses the total page count.	Read only	None
PageNumber	Long. Gets the page number.	Read only	None
Parent	"PageGenerator Object". Gets reference to the parent object.	Read only	None

Page Object Methods

The following methods are discussed in this section:

- "RenderEPF Method (Page Object)" on page 93
- "RenderHTML Method (Page Object)" on page 94

RenderEPF Method (Page Object)

The RenderEPF method returns a variant that contains EPF data for the report page. This method can be invoked only when in formatting idle mode.

Syntax

```
Function RenderEPF (resultType As CRRenderResultType)
```

Parameter

Parameter	Description
ResultType	"CRRenderResultType". Specifies whether the page will be rendered using strings or arrays. See Remarks below.

Returns

Returns the EPF stream.

Remarks

Currently only arrays are supported.

RenderHTML Method (Page Object)

The RenderHTML method returns a variant that contains HTML data for the report page. This method can be invoked only when in formatting idle mode.

Syntax

```
Function RenderHTML (includeDrillDownLinks As Boolean,  
    pageStyle As CRHTMLPageStyle, toolbarStyle As CRHTMLToolbarStyle,  
    baseURL As String, resultType As CRRenderResultType)
```

Parameters

Parameter	Description
includeDrillDownLinks	Indicates whether or not the HTML page will include hyperlinks for drilling-down on summary data.
pageStyle	"CRHTMLPageStyle". Specifies the style of the HTML page to be rendered.
toolbarStyle	"CRHTMLToolbarStyle". Bitwise constant specifies the style of the toolbar to be used. Constants can be XOR'd.
baseURL	The URL used to access the report when it is first generated.
resultType	"CRRenderResultType". Specifies whether the page will be rendered using strings or arrays. See Remarks below.

Returns

Returns the HTML stream.

Remarks

Currently only arrays are supported.

PageEngine Object

Use the PageEngine object when designing web sites using Active Server Pages, the Crystal Report Engine Automation Server, and the Crystal Design-Time ActiveX Control. Unless you are experienced with the Crystal Report Engine Object Library, you should allow the Crystal Design-Time ActiveX Control to generate VBScript code in your Active Server Pages that controls the Page Engine objects.

The Page Engine generates pages of a report on the web server and sends the pages to client web browsers as they are requested. For example, when a user first requests a report, only the first page is sent to the web browser. If the user pages forward or backward in the report, or requests a specific page, only that page is sent. This limits the resources required by the web server and reduces download time for the client browser.

PageEngine Object Properties

Property	Description	Read/Write	Restriction in event handler
ImageOptions	"CRImageType". Gets or sets the image type for EPF format.	Read/Write	Can be written only when formatting idle.
Parent	"Report Object". Gets reference to the parent object.	Read only	None
Placeholder Options	"CRPlaceholderType". Gets or sets the EPF place holder options.	Read/Write	Can be written only when formatting idle.
ValueFormat Options	"CRValueFormatType". Gets or sets the EPF value format options.	Read/Write	Can be written only when formatting idle.

PageEngine Object Methods

The following methods are discussed in this section:

- "CreatePageGenerator Method (PageEngine Object)" on page 95
- "RenderTotallerETF Method (PageEngine Object)" on page 96
- "RenderTotallerHTML Method (PageEngine Object)" on page 96

CreatePageGenerator Method (PageEngine Object)

The CreatePageGenerator method returns a "PageGenerator Object", allowing you to get pages from the view of the report, specified by the GroupPath parameter. This method can be invoked only when in formatting idle mode.

Syntax

Function CreatePageGenerator (GroupPath, [DrillDownLevel]) As PageGenerator

Parameter

Parameter	Description
GroupPath	Specifies an integer array that represents the group path. An empty array would represent the entire report, an array (0, 1) would represent a drill down on the second group member within the first group member where 0 = first group in Group #1 and 1 = second group in Group #2.
[DrillDownLevel]	Reserved. Do not use.

Returns

Returns a "PageGenerator Object".

RenderTotallerETF Method (PageEngine Object)

The RenderTotallerETF method returns a variant that contains ETF data for the Group Tree. This method can be invoked only when in formatting idle mode.

Syntax

```
Function RenderTotallerETF (rootGroupPath, startingChildNumber As Long,
    pastRootLevels As Integer, maxNodeCount,
    resultType As CRRRenderResultType)
```

Parameters

Parameter	Description
rootGroupPath	Specifies an integer array that represents the group path. An empty array would represent the entire report, an array (0, 1) would represent a drill down on the second group member within the first group member where 0 = first group in Group #1 and 1 = second group in Group #2.
startingChildNumber	The start child number to display.
pastRootLevels	Past root levels.
maxNodeCount	The maximum number of nodes for each group level in the Group Tree.
resultType	"CRRRenderResultType". Specifies whether the page will be rendered using strings or arrays. See Remarks below.

Returns

Returns the Totaller ETF stream.

Remarks

Currently only arrays are supported.

RenderTotallerHTML Method (PageEngine Object)

The RenderTotallerHTML method returns a variant that contains HTML data for the Group Tree. This method can be invoked only when in formatting idle mode.

Syntax

```
Function RenderTotallerHTML (rootGroupPath, startingChildNumber As Long,
    pastRootLevels As Integer, maxNodeCount, openGroupPath,
    includeDrillDownLinks As Boolean, baseUrl As String,
    resultType As CRRRenderResultType)
```

Parameters

Parameter	Description
rootGroupPath	Specifies the base URL string.
startingChildNumber	The start child number to display.
pastRootLevels	A value indicating the number of past root levels.
maxNodeCount	The maximum number of nodes to display for each group level in the Group Tree.
openGroupPath	An array of groups to be opened in the report.
includeDrillDownLinks	Indicates whether or not drill down hyperlinks are generated for summary values in the report.
baseURL	The URL used to access the report when it is first generated.
resultType	" CRRenderResultType ". Specifies whether the page will be rendered using strings or arrays. See Remarks below.

Returns

Returns the HTML stream.

Remarks

Currently only arrays are supported.

PageGenerator Object

The PageGenerator Object is part of the Page Engine. Use the Page Engine when designing web sites using Active Server Pages, the Crystal Report Engine Automation Server, and the Crystal Design-Time ActiveX Control. Unless you are experienced with the Crystal Report Engine Object Library, you should allow the Crystal Design-Time ActiveX Control to generate VBScript code in your Active Server Pages that controls the Page Engine objects.

The Page Engine generates pages of a report on the web server and sends the pages to client web browsers as they are requested. For example, when a user first requests a report, only the first page is sent to the web browser. If the user pages forward or backward in the report, or requests a specific page, only that page is sent. This limits the resources required by the web server and reduces download time for the client browser. A PageGenerator object generates "**Page Object**" as they are requested and allows options for manipulating the report as a whole.

PageGenerator Object Properties

Property	Description	Read/Write	Restriction in event handler
ContainingGroupName	String. Gets containing group name for out of place subreport view.	Read only	None
ContainingGroupPath	Variant. Gets containing group path for out of place subreport view.	Read only	None
ContainingPageNumber	Long. Gets containing page number for out of place subreport view.	Read only	None
DrillDownLevel	Ignore. This property is currently reserved.	Read only	None
GroupName	String. Gets the group name for drill down on graph view.	Read only	None
GroupPath	Variant. Gets the GroupPath parameter set by "CreatePageGenerator Method (PageEngine Object)" .	Read only	None
Pages	"Pages Collection" . Gets the collection of pages for a specified view.	Read only	None
Parent	"PageEngine Object" . Gets reference to the parent object.	Read only	None
ReportName	String. Gets the report name for drill down on out of place subreport view.	Read only	None
xOffset	Long. Gets the object x-offset.	Read only	None
yOffset	Long. Gets the object y-offset.	Read only	None

PageGenerator Object Methods

The following methods are discussed in this section:

- ["CreateSubreportPageGenerator Method \(PageGenerator Object\)"](#) on page 99
- ["DrillOnGraph Method \(PageGenerator Object\)"](#) on page 99
- ["DrillOnMap Method \(PageGenerator Object\)"](#) on page 100
- ["DrillOnSubreport Method \(PageGenerator Object\)"](#) on page 100
- ["Export Method \(PageGenerator Object\)"](#) on page 101
- ["FindText Method \(PageGenerator Object\)"](#) on page 101
- ["GetPageNumberForGroup Method \(PageGenerator Object\)"](#) on page 101
- ["RenderTotalerETF Method \(PageGenerator Object\)"](#) on page 102
- ["RenderTotalerHTML Method \(PageGenerator Object\)"](#) on page 103

CreateSubreportPageGenerator Method (PageGenerator Object)

Use the CreateSubreportPageGenerator method to create a page generator that contains the subreport view context.

Syntax

```
Function CreateSubreportPageGenerator (GroupPath, _  
[DrillDownLevel]) As PageGenerator
```

Parameters

Parameter	Description
GroupPath	Specifies an integer array that represents the group path. An empty array would represent the entire report, an array (0, 1) would represent a drill down on the second group member within the first group member where 0 = first group in Group #1 and 1 = second group in Group #2.
[DrillDownLevel]	Reserved. Do not use.

DrillOnGraph Method (PageGenerator Object)

The DrillOnGraph method creates a new “PageGenerator Object” that results from drilling down on the specified point in a graph on the given page. This method can be invoked only when in formatting idle mode.

Syntax

```
Function DrillOnGraph (PageNumber As Long, xOffset As Long, _  
yOffset As Long) As PageGenerator
```

Parameters

Parameter	Description
PageNumber	Specifies the page number.
xOffset	Specifies the X coordinate on page, in twips, where the drill down occurred.
yYOffset	Specifies the Y coordinate on page, in twips, where the drill down occurred.

Returns

Returns a “PageGenerator Object”.

DrillOnMap Method (PageGenerator Object)

The DrillOnMap method creates a new “PageGenerator Object” that results from drilling down on the specified point in a map on the given page. This method can be invoked only when in formatting idle mode.

Syntax

```
Function DrillOnMap (PageNumber As Long, xOffset As Long, _
    yOffset As Long) As PageGenerator
```

Parameters

Parameter	Description
PageNumber	Specifies the page number.
xOffset	Specifies the X coordinate on page, in twips, where the drill down occurred.
yOffset	Specifies the Y coordinate on page, in twips, where the drill down occurred.

Returns

Returns a “PageGenerator Object”.

DrillOnSubreport Method (PageGenerator Object)

The DrillOnSubreport method creates a new “PageGenerator Object” that results from drilling down on the specified point in a real-time subreport on the given page. This method can be invoked only when in formatting idle mode.

Syntax

```
Function DrillOnSubreport (PageNumber As Long, xOffset As Long, _
    yOffset As Long) As PageGenerator
```

Parameters

Parameter	Description
PageNumber	Specifies the page number.
xOffset	Specifies the X coordinate on page, in twips, where the drill down occurred.
yOffset	Specifies the Y coordinate on page, in twips, where the drill down occurred.

Returns

Returns a “PageGenerator Object”.

Export Method (PageGenerator Object)

Use the Export method to obtain an export stream.

Syntax

```
Function Export (resultType As CRRenderResultType)
```

Parameter

Parameter	Description
resultType	"CRRenderResultType". Specifies the result type.

Returns

Returns the export data stream.

FindText Method (PageGenerator Object)

Use the FindText method to search for a text string in the specified direction starting on the specified page in the current drill down view.

Syntax

```
Function FindText (Text As String, direction As CRSearchDirection, _  
pPageNumber) As Boolean
```

Parameters

Parameter	Description
Text	Specifies the text string that you want to search for.
direction	"CRSearchDirection". Specifies the direction that you want to search.
pPageNumber	Specifies the page on which you want to start the search.

Returns

- TRUE if the specified text string is found.
- FALSE if the specified text string is not found.

GetPageNumberForGroup Method (PageGenerator Object)

The GetPageNumberForGroup method returns the page number on which the specified group starts. This method can be invoked only when in formatting idle mode.

Syntax

```
Function GetPageNumberForGroup (GroupPath) As Long
```

Parameters

Parameter	Description
GroupPath	Specifies an integer array that represents the group path. An empty array would represent the entire report, an array (0, 1) would represent a drill down on the second group member within the first group member where 0 = first group in Group #1 and 1 = second group in Group #2.

Returns

Returns the page number on which the specified group starts.

RenderTotallerETF Method (PageGenerator Object)

The RenderTotallerETF method returns a variant that contains ETF data for the Group Tree. This method can be invoked only when in formatting idle mode.

Syntax

```
Function RenderTotallerETF (rootGroupPath, startingChildNumber As Long, _
    pastRootLevels As Integer, maxNodeCount, _
    resultType As CRRenderResultType)
```

Parameters

Parameter	Description
rootGroupPath	Specifies an integer array that represents the root group path. An empty array would represent the entire report, an array (0, 1) would represent a drill down on the second group member within the first group member where 0 = first group in Group #1 and 1 = second group in Group #2.
startingChildNumber	The starting child level to display the report grouping at.
pastRootLevels	Specifies the number of past root levels.
maxNodeCount	The maximum number of nodes for each group level in the Group Tree.
resultType	" CRRenderResultType ". Specifies whether the page will be rendered using strings or arrays. See Remarks below.

Returns

Returns the Totaller ETF stream.

Remarks

Currently only arrays are supported.

RenderTotallerHTML Method (PageGenerator Object)

The RenderTotallerHTML method returns a variant that contains HTML data for the Group Tree. This method can be invoked only when in formatting idle mode.

Syntax

```
Function RenderTotallerHTML (rootGroupPath, startingChildNumber As Long, _
    pastRootLevels As Integer, maxNodeCount, openGroupPath, _
    includeDrillDownLinks As Boolean, baseURL As String, _
    resultType As CRRRenderResultType)
```

Parameters

Parameter	Description
rootGroupPath	Specifies an integer array that represents the root group path. An empty array would represent the entire report, an array (0, 1) would represent a drill down on the second group member within the first group member where 0 = first group in Group #1 and 1 = second group in Group #2.
startingChild Number	The starting child level to display the report grouping at.
pastRootLevels	A value indicating the number of past root levels.
maxNodeCount	The maximum number of nodes to display for each group level in the Group Tree.
openGroupPath	An array of groups to be opened in the report.
includeDrill DownLinks	Indicates whether or not drill down hyperlinks are generated for summary values in the report.
baseURL	The URL used to access the report when it is first generated.
resultType	"CRRRenderResultType". Specifies whether the page will be rendered using strings or arrays. See Remarks below.

Returns

Returns the HTML stream.

Remarks

Currently only arrays are supported.

Pages Collection

The Pages Collection is part of the Page Engine. Use the Page Engine when designing web sites using Active Server Pages, the Crystal Report Engine Automation Server, and the Crystal Design-Time ActiveX Control. Unless you are experienced with the Crystal Report Engine Object Library, you should allow the Crystal Design-Time ActiveX Control to generate VBScript code in your Active Server Pages that controls the Page Engine objects.

The Page Engine generates pages of a report on the web server and sends the pages to client web browsers as they are requested. For example, when a user first requests a report, only the first page is sent to the web browser. If the user pages forward or backward in the report, or requests a specific page, only that page is sent. This limits the resources required by the web server and reduces download time for the client browser.

The Pages Collection is a collection of “Page Object”. Access a specific Page Object in the collection using the Item property.

Pages Collection Properties

Property	Description	Read/Write	Restriction in event handler
Count	Long. Gets the number of page objects in the collection.	Read only	None
Item (index As Long)	“Page Object”. Gets the collection item based on the 1-based index number.	Read only	None
Parent	“PageGenerator Object”. Gets reference to the parent object.	Read only	None

Remarks

Item is a default property. You can reference a page directly, for example, Pages(1).

ParameterFieldDefinition Object

The ParameterFieldDefinition Object represents a parameter field in the report. This object provides properties and methods for retrieving information and setting options for a parameter field in your report (that is, current value, default value). A ParameterFieldDefinition Object is obtain from the Field property of the “FieldMappingData Object”, when the specified field is a parameter field or from the “ParameterFieldDefinitions Collection” Object under Report.

ParameterFieldDefinition Object Properties

Property	Description	Read/Write	Restriction in event handler
DisallowEditing	Boolean. Gets or sets the disallowing editing option.	Read/Write	Can be written only when formatting idle.
DiscreteOrRangeKind	“CRDiscreteOrRangeKind”. Gets or sets the parameter value kind (discrete and/or range).	Read/Write	Can be written only when formatting idle.

Property	Description	Read/Write	Restriction in event handler
EditMask	String. Gets or sets edit mask for string parameter. See Remarks below for additional information.	Read/Write	Can be written only when formatting idle.
EnableExclusiveGroup	Boolean. Works in conjunction with property PlaceInGroup. If EnableExclusiveGroup is TRUE, a Bool parameter group can have only one value set to TRUE; if FALSE, then the group can have multiple values set to TRUE.	Read/Write	Can be written only when formatting idle.
EnableMultipleValues	Boolean. Gets or sets the allow multiple values option.	Read/Write	Can be written only when formatting idle.
EnableNullValue	Boolean. Gets or sets the value nullable option for Stored Procedure parameters.	Read/Write	Can be written only when formatting idle.
EnableRangeLimit	Boolean. Gets or sets the option which specifies if this parameter field value should be in the specified range.	Read/Write	Can be written only when formatting idle.
EnableShowDescriptionOnly	Boolean. Gets or sets the show description for pick list option.	Read/Write	Can be written only when formatting idle.
EnableSortBasedOnDesc	Boolean. Gets or sets the sort based on description in pick list option.	Read/Write	Can be written only when formatting idle.
GroupNumber	Integer. Gets or sets boolean group number.	Read/Write	Can be written only when formatting idle.
IsCurrentValueSet	Boolean. Gets whether the current value has been set.	Read only	None
IsDefaultValueSet	Boolean. Gets whether the default value has been set.	Read only	None
Kind	"CRFieldKind". Gets which kind of field (database, summary, formula, etc.).	Read only	None
MaximumValue	Variant. Gets or sets the maximum value. See Remarks below for additional comments.	Read/Write	Can be written only when formatting idle.
MinimumValue	Variant. Gets or sets minimum value. See Remarks below for additional comments.	Read/Write	Can be written only when formatting idle.
Name	String. Gets the unique formula name of the parameter field as it appears in the Parameter Field list (RDC DataView).	Read only	None
NeedsCurrentValue	Boolean. Gets whether the field needs a current value.	Read only	None
NextValue	Variant. Gets the next field value.	Read only	Can be read only when top-level Report object is formatting active.

Property	Description	Read/Write	Restriction in event handler
NthValue Description (index As Integer)	String. Gets or sets nth value description.	Read/Write	Can be written only when formatting idle.
NumberOfBytes	Integer. Gets the number of bytes required to store the field data in memory.	Read only	None
NumberOfCurrent Ranges	Integer. Gets total number of current ranges.	Read only	None
NumberOfCurrent Values	Integer. Gets the total number of current values.	Read only	None
NumberOfDefault Values	Integer. Gets the total number of default values.	Read only	None
ParameterField Name	String. Gets the name of the parameter field as it is displayed (referenced) in the report (RDC DataView).	Read only	None
ParameterType	"CRParameterFieldType". Gets parameter type.	Read only	None
Parent	"Report Object". Gets reference to the parent object.	Read only	None
PickListSort Method	"CRParameterPickListSortMethod". Gets or sets the sort method in pick list option.	Read/Write	Can be written only when formatting idle.
PlaceInGroup	Boolean. Gets or sets, when prompting for values, whether a Boolean parameter field should appear as part of a Boolean parameter group or individually. Used in conjunction with Boolean property EnableExclusiveGroup.	Read/Write	Can be written only when formatting idle.
PreviousValue	Variant. Gets the field previous value.	Read only	Can be read only when top-level Report object is formatting active.
Prompt	String. Gets or sets the parameter field prompting string.	Read/Write	None
ReportName	String. Gets the report name the parameter field is in. If it is a main report, the ReportName is empty.	Read only	None
Value	Variant. Gets the current value of the field.	Read only	Can be read only when top-level Report object is formatting active.
ValueType	"CRFieldValueType". Gets which type of value is found in the field.	Read only	None

Remarks

- Regarding property EditMask, please see the additional information available in Crystal Reports Online Help. Search for "Edit Parameter Field dialog box" in the SCR Online Help index.
- Regarding properties MaximumValue and MinimumValue, the following comments apply.
 - All parameter field types: EnableRangeLimit property must be set to TRUE for MaximumValue and Minimum Value properties to have an effect.
 - String parameter fields: The properties provide the maximum and minimum lengths of the string, not a value range.
 - Boolean parameter fields: The properties do not apply.
- Regarding properties EditMask, MaximumValue and MinimumValue, changing values associated with one or more of these properties may result in the loss of default values that do not fall within the scope of the new EditMask, MaximumValue or MinimumValue properties.

ParameterFieldDefinition Object Methods

The following methods are discussed in this section:

- "AddCurrentRange Method (ParameterFieldDefinition Object)" on page 107
- "AddCurrentValue Method (ParameterFieldDefinition Object)" on page 108
- "AddDefaultValue Method (ParameterFieldDefinition Object)" on page 108
- "ClearCurrentValueAndRange Method (ParameterFieldDefinition Object)" on page 109
- "DeleteNthDefaultValue Method (ParameterFieldDefinition Object)" on page 109
- "GetNthCurrentRange Method (ParameterFieldDefinition Object)" on page 109
- "GetNthCurrentValue Method (ParameterFieldDefinition Object)" on page 110
- "GetNthDefaultValue Method (ParameterFieldDefinition Object)" on page 110
- "SetNthDefaultValue Method (ParameterFieldDefinition Object)" on page 110

AddCurrentRange Method (ParameterFieldDefinition Object)

The AddCurrentRange method adds the current parameter range to the specified parameter field of a report. When this method is used, property DiscreteOrRangeKind must be crRangeValue.

Syntax

```
Sub AddCurrentRange (start, end, rangeInfo As CRRangeInfo)
```

Parameters

Parameter	Description
start	Sets start of the value range.
end	Sets end of the value range.
rangeInfo	"CRRangeInfo". Use this bitwise value to indicate whether the upper and/or lower bound of the range should be included.

AddCurrentValue Method (ParameterFieldDefinition Object)

The AddCurrentValue method adds a value to the specified parameter field of a report. When this method is used, property DiscreteOrRangeKind must be crDiscreteValue.

Syntax

```
Sub AddCurrentValue (CurrentValue)
```

Parameter

Parameter	Description
CurrentValue	Specifies the current value to be added.

AddDefaultValue Method (ParameterFieldDefinition Object)

The AddDefaultValue method adds a value to the group of default values for a specified parameter in a report. The default value set should fall within the scope of properties EditMask and MaximumValue and MinimumValue, if set. Use this method instead of SetDefaultValue if the parameter field allows multiple values.

Syntax

```
Sub AddDefaultValue (DefaultValue)
```

Parameter

Parameter	Description
DefaultValue	Specifies the default value to be added.

ClearCurrentValueAndRange Method(ParameterFieldDefinition Object)

The ClearCurrentValueAndRange method clears the specified parameter field of all current values and ranges.

Syntax

```
Sub ClearCurrentValueAndRange ( )
```

DeleteNthDefaultValue Method (ParameterFieldDefinition Object)

The DeleteNthDefaultValue method deletes the nth default value of the parameter field.

Syntax

```
Sub DeleteNthDefaultValue(index As Integer)
```

Parameter

Parameter	Description
index	Index of the value to be deleted.

GetNthCurrentRange Method (ParameterFieldDefinition Object)

The GetNthCurrentRange method retrieves a value range from the specified parameter field in a report.

Syntax

```
Sub GetNthCurrentRange (index As Integer, pStart, pEnd, _  
    pRangeInfo As CRRangeInfo)
```

Parameters

Parameter	Description
index	Index of the value range to be retrieved.
pStart	Sets start of the value range.
pEnd	Sets end of the value range.
pRangeInfo	"CRRangeInfo". Use this bitwise value to indicate whether the upper and/or lower bound of the range should be included.

GetNthCurrentValue Method (ParameterFieldDefinition Object)

The GetNthCurrentValue method returns a value from the specified parameter field of a report.

Syntax

```
Function GetNthCurrentValue (index As Integer)
```

Parameter

Parameter	Description
index	Index number of the current value to be retrieved.

Returns

Returns the current value specified by the index parameter.

GetNthDefaultValue Method (ParameterFieldDefinition Object)

Get NthDefaultValue method retrieves a default value for a specified parameter field in a report.

Syntax

```
Function GetNthDefaultValue (index As Integer)
```

Parameter

Parameter	Description
index	The index number of the default value to be retrieved.

Returns

Returns the default value specified by the index parameter.

SetNthDefaultValue Method (ParameterFieldDefinition Object)

The SetNthDefaultValue method sets a default value for a specified parameter field in a report. The default value set should fall within range and property EditMask if a string field, if set.

Syntax

```
Sub SetNthDefaultValue (index As Integer, nthDefaultValue)
```

Parameters

Parameter	Description
index	The index number of the default value to be set.
nthDefaultValue	Specifies the default value that you want to set for parameter field.

ParameterFieldDefinitions Collection

The ParameterFieldDefinitions Collection is a collection of parameter fields in the report. If the report contains any subreports, parameter fields in the subreports will also be included in the collection. Access a specific ParameterFieldDefinition.

ParameterFieldDefinitions Collection Properties

Property	Description	Read/Write	Restriction in event handler
Count	Long. Gets the number of parameter fields in the collection.	Read only	None
Item (index As Long)	"ParameterFieldDefinition Object". Gets the item in the Collection specified by parameter index.	Read only	None
Parent	"Report Object". Gets reference to the parent object.	Read only	None

Remarks

Item is a default property. You can reference a parameter field directly, for example, ParameterFieldDefinitions(1).

ParameterFieldDefinitions Collection Method

The following method is discussed in this section:

- "GetItemByName Method (ParameterFieldDefinitions Collection)" on page 111

GetItemByName Method (ParameterFieldDefinitions Collection)

Use the GetItemByName method to retrieve a DatabaseFieldDefinition object by name.

Syntax

```
Sub GetItemByName(name As String, parameterFieldDefinition)
```

Parameters

Parameter	Description
name	The item's unique name index.
parameterFieldDefinition	A ParameterFieldDefinition object to hold the retrieved item.

ParameterValue Object

The ParameterValue Object represents the initial value of a stored procedure parameter that will eventually be passed to the “AddStoredProcedure Method (DatabaseTables Collection)”.

Note: The Prog Id CrystalRuntime.ParameterValue can be used with the CreateObject function in Visual Basic to create an instance of a ParameterValue object at runtime.

```
dim pv as Variant
set pv = CreateObject("CrystalRuntime.ParameterValue")
```

ParameterValue Object Properties

Property	Description	Read/Write	Restriction in event handler
StartValue	Gets or sets the start value of the range.	Read/Write	None
EndValue	Gets or sets the end value of the range.	Read/Write	None
RangeValue	Boolean. True if it is a range value.	Read/Write	None
RangeInfo	“CRRangeInfo”. Use this bitwise value to indicate whether the upper and/or lower bound of the range should be included.	Read/Write	None

ParameterValues Collection

The ParameterValues Collection is a collection of ParameterValue Objects that will eventually be passed to the “AddStoredProcedure Method (DatabaseTables Collection)”.

Note: The Prog Id CrystalRuntime.ParameterValues can be used with the CreateObject function in Visual Basic to create an instance of a ParameterValues collection at runtime.

```
dim pv as Variant
set pv = CreateObject("CrystalRuntime.ParameterValues")
```

ParameterValues Collection Properties

Property	Description	Read/Write	Restriction in event handler
Count	Long. Gets the number of parameter fields in the collection.	Read only	None
Item (index As Long)	“ParameterValue Object”. Gets the item in the Collection specified by parameter index.	Read only	None

ParameterValues Collection Methods

The following methods are discussed in this section:

- “Add Method (ParameterValues Collection)” on page 113
- “Delete Method (ParameterValues Collection)” on page 113

Add Method (ParameterValues Collection)

Use the Add method to add a **ParameterValue Object** to the ParameterValues Collection.

Syntax

```
Sub Add(pValue As ParameterValue)
```

Parameter

Parameter	Description
pValue	The ParameterValue object to add to the Collection.

Delete Method (ParameterValues Collection)

Use the Delete method to remove a ParameterValue Object from the Collection.

Syntax

```
Sub Delete (index)
```

Parameter

Parameter	Description
index	Specifies the index number of the item that you want to delete from the Collection.

ParameterValueInfo Object

The ParameterValueInfo Object is comprised of ParameterValues Collections. ParameterValueInfo Objects are eventually passed to the **AddStoredProcedure Method (DatabaseTables Collection)**.

Note: The Prog Id CrystalRuntime.ParameterValueInfo can be used with the CreateObject function in Visual Basic to create an instance of a ParameterValueInfo object at runtime.

```
dim pv as Variant
set pv = CreateObject("CrystalRuntime.ParameterValueInfo")
```

ParameterValueInfo Object Properties

Property	Description	Read/Write	Restriction in event handler
ParameterValues	Gets or sets the value of a parameter.	Read / Write	None
ParameterName	Gets or sets the name of a parameter.	Read / Write	None

ParameterValueInfos Collection

ParameterValueInfos is a collection of ParameterValueInfo objects which is passed to the [“AddStoredProcedure Method \(DatabaseTables Collection\)”](#).

Note: The Prog Id CrystalRuntime.ParameterValueInfos can be used with the CreateObject function in Visual Basic to create an instance of a ParameterValueInfos object at runtime.

```
dim pv as Variant
set pv = CreateObject("CrystalRuntime.ParameterValueInfos")
```

ParameterValueInfos Collection Properties

Property	Description	Read/Write	Restriction in event handler
Count	Long. Gets the number of parameter fields in the collection.	Read only	None
Item (index As Long)	“Parameter Value Object” . Gets the item in the Collection specified by parameter index.	Read only	None

ParameterValueInfos Collection Methods

The following methods are discussed in this section:

- [“Add Method \(ParameterValueInfos Collection\)”](#) on page 114
- [“Delete Method \(ParameterValueInfos Collection\)”](#) on page 115

Add Method (ParameterValueInfos Collection)

Use the Add method to add a [“Parameter ValueInfo Object”](#) to the ParameterValueInfos Collection.

Syntax

```
Sub Add(pValueInfo As ParameterValueInfo)
```

Parameter

Parameter	Description
pValueInfo	The ParameterValueInfo object to add to the Collection.

Returns

Returns the ParameterValueInfo Object added to the Collection.

Delete Method (ParameterValueInfos Collection)

Use the Delete method to remove a ParameterValueInfo Object from the Collection.

Syntax

Sub Delete (index)

Parameter

Parameter	Description
index	Specifies the index number of the item that you want to delete from the Collection.

PrintingStatus Object

The PrintingStatus Object provides properties for retrieving information and setting options for the printing status of a report (for example, number of pages, latest page to be printed). A PrintingStatus Object is obtained from the PrintingStatus property of the *“Report Object”*.

PrintingStatus Object Properties

Property	Description	Read/Write	Restriction in event handler
NumberOfPages	Long. Gets the total number of pages in the report.	Read only	None
NumberOfRecordPrinted	Long. Gets the number of records printed.	Read only	None
NumberOfRecordRead	Long. Gets the number of records read.	Read only	None
NumberOfRecordSelected	Long. Gets the number of records selected.	Read only	None
Parent	<i>“Report Object”</i> . Gets reference to the parent object.	Read only	None
Progress	<i>“CRPrintingProgress”</i> . Gets the printing progress.	Read only	None

Report Object

A report corresponds to a print job in the Crystal Report Engine. When the report object is destroyed, or goes out of focus, it closes the print job. It holds on to “Application Object”. When a Report Object gets destroyed, it releases the application.

Access to the Report Object is dependent on the object variable you create. If the object variable goes out of scope, you will lose access to the Report Object and, therefore, the report. You may want to declare your Report Object variable as Global.

Report Object Properties

Property	Description	Read/Write	Restriction in event handler
Application	“Application Object”. Gets reference to the Application Object that the Report Object is associated with.	Read only	None
ApplicationName	String. Gets or sets the application name.	Read/Write	None
Areas	“Areas Collection”. Gets reference to the Areas Collection, a collection of all the areas in the report which can be indexed by a number or by a string, such as “RH”, “GF1”. The areas are in the same order as on the Crystal Reports Design Tab. For Example: RH, PH, GH1,...GHn, D, GFn,...GF1, RF, PF. The abbreviations for areas are case sensitive.	Read only	None
BottomMargin	Long. Gets or sets the page bottom margin, in twips.	Read/Write	Can be written only when formatting idle.
CanPerformGroupingOnServer	Boolean. Gets whether the report can perform grouping on the server.	Read only	None
CaseInsensitiveSQLData	Boolean. Gets or sets the report option that indicates whether the SQL data used in the report becomes case sensitive.	Read/Write	Can be written only when formatting idle.
ConvertDateTimeType	“CRCommonFieldFormatConditionFormulaType”. Gets or sets the report option that specifies the format to be converted for date/time fields.	Read/Write	Can be written only when formatting idle.
ConvertNullFieldToDefault	Boolean. Gets or sets the report option that indicates whether to convert any null values to the database field default.	Read/Write	Can be written only when formatting idle.

Database	"Database Object". Gets reference to the Database Object which represents the database used in the report.	Read only	None
DisplayProgress Dialog	Boolean. Enable or disable the progress dialog.	Read/Write	Can be written only when formatting idle.
DriverName	String. Gets the printer driver name used by the current report. Gets an empty string if default printer is used.	Read only	Can be written only when formatting idle.
EnableAsync Query	Boolean. Gets or sets the enable AsyncQuery.	Read/Write	Can be written only when formatting idle.
EnableGenerating DataForHidden Object	Boolean. Gets or sets the Enable Generating Data For Hidden Object option.	Read/Write	Can be written only when formatting idle.
EnableParameter Prompting	Boolean. Gets or sets the prompting for parameter fields option.	Read/Write	Can be written only when formatting idle.
EnablePerform Queries Asynchronously	Boolean. Gets or sets the perform queries asynchronously option.	Read/Write	Can be written only when formatting idle.
EnableSelect DistinctRecords	Boolean. Gets or sets the select distinct records option	Read/Write	Can be written only when formatting idle.
ExportOptions	"ExportOptions Object". Gets reference to ExportOptions Object for the report.	Read only	None
FieldMapping Type	"CRFieldMappingType". Gets or sets the field mapping type.	Read/Write	Can be written only when formatting idle.
FormulaFields	"FormulaFieldDefinitions Collection". Gets reference to Collection of all the named FormulaFieldDefinitions defined in the Report.	Read only	None
FormulaSyntax	"CRFontColorConditionFormulaType". Gets or sets report formula syntax.	Read/Write	Can be written only when formatting idle.
GroupNameFields	"GroupNameFieldDefinitions Collection". Gets reference to a collection of all the group name fields defined in the report.	Read only	None
GroupSelection Formula	String. Gets or sets the group selection formula.	Read/Write	Can be written only when formatting idle.
GroupSortFields	"SortFields Collection". Gets reference to group sort field collection.	Read only	None
HasSavedData	Boolean. Gets whether the report has data saved in memory.	Read only	None
KeywordsIn Report	String. Gets or sets the keywords in the report.	Read/Write	Can be written only when formatting idle.
Kind	"CRReportKind". Gets what kind of report.	Read only	None

LastGetFormula Syntax	" CRFontColorConditionFormulaType ". Gets the formula syntax of the last formula text returned.	Read only	None
LeftMargin	Long. Gets or sets the page left margin, in twips.	Read/Write	Can be written only when formatting idle.
NeedUpdated Pages	Gets whether the user needs to update pages for display due to changes in the report.	Read only	None
NumberOfGroup	Long. Gets the number of groups in the report.	Read only	None
PageEngine	" PageEngine Object ". Gets reference to the PageEngine object.	Read only	None
PaperOrientation	" CROpenReportMethod ". Gets or sets the current printer paper orientation. For the default printer, crDefaultPaperOrientation is returned.	Read/Write	Can be written only when formatting idle.
PaperSize	" CRPaperSize ". Gets or sets the current printer paper size. For the default printer, crDefaultPaperSize is returned.	Read/Write	Can be read or written only when formatting idle.
PaperSource	" CRPaperSource ". Gets or sets the current printer paper source.	Read/Write	Can be written only when formatting idle.
ParameterFields	" ParameterFieldDefinitions Collection ". Gets reference to the collection of all the ParameterFieldDefinitions defined in the report. This property will return parameter fields found in the main report as well as any subreports included in the report (for example, if the main report has 3 parameters and a subreport included within the report has an additional 2 parameters, the number of parameter fields in the collection returned by Report.ParameterFields would be 5).	Read only	None
Parent	" Report Object ". Gets reference to the parent object for subreports. (NULL for main report).	Read only	None
PerformGrouping OnServer	Boolean. Gets or sets the performing grouping on server option.	Read/Write	Can be written only when formatting idle.
PortName	String. Gets the printer port name used by the current report. Gets an empty string if the default printer is used.	Read only	None
PrintDate	Date. Gets or sets the print date for the report. By default, the current date will be used.	Read/Write	Can be written only when formatting idle.

PrinterDuplex	"CRPrinterDuplexType". Gets or sets the current printer duplex option.	Read/Write	Can be written only when formatting idle.
PrinterName	String. Gets the printer name used by the report. Gets an empty string if the default printer is used.	Read only	None
PrintingStatus	"PrintingStatus Object". Gets PrintingStatus Object for the report.	Read only	None
RecordSelectionFormula	String. Gets or sets record selection formula.	Read/Write	Can be written only when formatting idle.
RecordSortFields	"SortFields Collection". Gets a collection of record sort fields.	Read only	None
ReportAlerts	"ReportAlerts Collection". Gets reference to Collection of all the named Report Alerts defined in the Report.	Read only	None
ReportAuthor	String. Gets or sets the report author.	Read/Write	Can be written only when formatting idle.
ReportComments	String. Gets or sets report comments.	Read/Write	Can be written only when formatting idle.
ReportSubject	String. Gets or sets the report subject.	Read/Write	Can be written only when formatting idle.
ReportTemplate	String. Gets or sets the report template.	Read/Write	Can be written only when formatting idle.
ReportTitle	String. Gets or sets the report title.	Read/Write	Can be written only when formatting idle.
RightMargin	Long. Gets or sets the page right margin, in twips.	Read/Write	Can be written only when formatting idle.
RunningTotalFields	"RunningTotalFieldDefinitions Collection". Gets running total fields collection.	Read only	None
SavePreviewPicture	Boolean. Gets or sets save preview picture with report option.	Read/Write	None
Sections	"Sections Collection". Gets collection of all the sections in the report.	Read only	None
SQLExpressionFields	"SQLExpressionFieldDefinitions Collection". Gets SQL expression field collection.	Read only	None
SQLQueryString	String. Gets or sets SQL query string.	Read/Write	Can be written only when formatting idle.
SummaryFields	"SummaryFieldDefinitions Collection". Gets collection for group and report summaries (cross-tab summaries not available using this property).	Read only	None

TopMargin	Long. Gets or sets the page top margin, in twips.	Read/Write	Can be written only when formatting idle.
UseIndexForSpeed	Boolean. Gets or sets the use index for speed during record selection report option.	Read/Write	Can be written only when formatting idle.
VerifyOnEveryPrint	Boolean. Gets or sets the report option that indicates whether to verify the database every time the report is printed.	Read/Write	Can be written only when formatting idle.

Report Object Methods

The following methods are discussed in this section:

- “AddGroup Method (Report Object)—RDC Object Model” on page 120
- “AddReportVariable Method (Report Object)” on page 121
- “AutoSetUnboundFieldSource Method (Report Object)” on page 121
- “CancelPrinting Method (Report Object)” on page 122
- “DeleteGroup Method (Report Object)” on page 122
- “DiscardSavedData Method (Report Object)” on page 122
- “Export Method (Report Object)” on page 122
- “GetNextRows Method (Report Object)” on page 123
- “GetReportVariableValue Method (Report Object)” on page 123
- “OpenSubreport Method (Report Object)” on page 124
- “PrinterSetup Method (Report Object)” on page 124
- “PrintOut Method (Report Object)” on page 124
- “PrintOutEx Method (Report Object)” on page 125
- “ReadRecords Method (Report Object)” on page 125
- “SaveAs Method (Report Object)” on page 126
- “SelectPrinter Method (Report Object)” on page 126
- “SetDialogParentWindow Method (Report Object)” on page 126
- “SetReportVariableValue Method (Report Object)” on page 127
- “SetUnboundFieldSource Method (FieldObject Object)” on page 73

AddGroup Method (Report Object)—RDC Object Model

The AddGroup Method adds a group to the report. ConditionField indicates the field for grouping, Condition indicates a change in a field value that generates a grouping, and SortDirection specifies the direction in which groups are sorted.

Syntax

```
Sub AddGroup (GroupNumber As Integer, pConditionField As Object, _
    Condition As CRGroupCondition, SortDirection As CRSortDirection)
```

Parameters

Parameter	Description
GroupNumber	Specifies the number of the group to be added (the position of the group in relation to existing groups). For example, to add a group to the first position, set GroupNumber=1.
pConditionField	Specifies the field to be grouped. The field can be a database field definition object or the field name.
Condition	"CRGroupCondition". Specifies CRGroupCondition indicating the grouping condition (for example, group on any value or group dates weekly).
SortDirection	"CRSortDirection". Specifies the sort direction for the group.

AddReportVariable Method (Report Object)

Use the AddReportVariable method to add a report variable to the report. This variable can then be used to provide a calculated field value in the Format Event. You can add as many report variables to your report as you need. Each report variable is identified by its name, which must be unique.

Syntax

```
Sub AddReportVariable (type As CRReportVariableValueType, _
    pName As String, [arraySize As Long], [reserved])
```

Parameters

Parameter	Description
type	"CRReportVariableValueType". Specifies the type of variable that you want to add to the report.
pName	Specifies the unique name for the report variable that you want to add.
[arraySize As Long]	Reserved. Do not use.
[reserved]	Reserved. Do not use.

AutoSetUnboundFieldSource Method (Report Object)

Use the AutoSetUnboundFieldSource method to automatically bind unbound report fields to database fields based on the unbound field's name or name and value type.

Syntax

```
Sub AutoSetUnboundFieldSource (matchType As CRBindingMatchType, _
    [bindSubReports])
```

Parameters

Parameter	Description
matchType	"CRBindingMatchType". Specifies whether to match name alone or name and data type.
[bindSubReports]	Specifies whether or not to bind subreport unbound fields.

CancelPrinting Method (Report Object)

Use the CancelPrinting method to cancel the printing of a report.

Syntax

```
Sub CancelPrinting ()
```

DeleteGroup Method (Report Object)

Use the DeleteGroup method to remove a group from a report.

Syntax

```
Sub DeleteGroup (GroupNumber As Integer)
```

Parameter

Parameter	Description
GroupNumber	Specifies the index number of the group that you want to delete from the report.

DiscardSavedData Method (Report Object)

Use the DiscardSavedData method to discard any saved data with the report before previewing. This method can be invoked only when in formatting idle mode.

Syntax

```
Sub DiscardSavedData ()
```

Export Method (Report Object)

Use the Export method to export reports to a format and destination specified with "ExportOptions Object".

Syntax

```
Sub Export ([promptUser])
```

Parameters

Parameter	Description
[promptUser]	Specifies Boolean value indicating if user should be prompted for export options. If you don't want to prompt the user, then you must set all necessary export options. The application will prompt automatically for any missing export options, even if promptUser = FALSE. Default value = TRUE (prompt user).

GetNextRows Method (Report Object)

Use the GetNextRows method to get the specified rowset.

Syntax

Function GetNextRows (startRowN As Long, pRowN As Long)

Parameters

Parameter	Description
startRowN	Specifies the row number to start the rowset on.
pRowN	Specifies the number of rows to return to the rowset.

Returns

Returns the specified rowset.

GetReportVariableValue Method (Report Object)

Use the GetReportVariableValue method to get the value of the specified uniquely named report variable. This call can only be made in the formatting active mode.

Syntax

Function GetReportVariableValue (pName As String)

Parameters

Parameter	Description
pName	Specifies the unique name of the report variable for which you want to get the value.

Returns

Returns value of the specified report variable.

OpenSubreport Method (Report Object)

The OpenSubreport method opens a subreport contained in the report and returns a Report Object corresponding to the named subreport.

Syntax

```
Function OpenSubreport (pSubreportName As String) As Report
```

Parameter

Parameter	Description
pSubreportName	Specifies the file name of the subreport to be opened.

Returns

Returns the specified subreport as a Report Object.

PrinterSetup Method (Report Object)

Use the PrinterSetup method to open the printer setup dialog box so that the user can change printers or printer settings for the report.

Syntax

```
Sub PrinterSetup (hWnd As Long)
```

Parameter

Parameter	Description
hWnd	Specifies the handle of the printer setup dialog box parent window. If you pass 0, the top level application window will be the parent.

PrintOut Method (Report Object)

Use the PrintOut method to print out the specified pages of the report to the printer selected using the “[SelectPrinter Method \(Report Object\)](#)”. If no printer is selected, the default printer specified in the report will be used. This method can be invoked only when in formatting idle mode.

Syntax

```
Sub PrintOut ([promptUser], [numberOfCopy], [collated], _  
[startPageN], [stopPageN])
```

Parameters

Parameter	Description
[promptUser]	Specifies Boolean value indicating if the user should be prompted for printer options.
[numberOfCopy]	Specifies the number of report copies you want printed.
[collated]	Specifies Boolean value specifying whether or not you want the report copies collated.
[startPageN]	Specifies the first page that you want printed.
[stopPageN]	Specifies the last page that you want printed.

PrintOutEx Method (Report Object)

Use the PrintOutEx method to export the specified pages of the report to the printer selected using the “[SelectPrinter Method \(Report Object\)](#)”. If no printer is selected, the default printer specified in the report will be used. This method can be invoked only when in formatting idle mode.

Syntax

```
Sub PrintOutEx ([promptUser], [numberOfCopy], [collated], [startPageN],  
[stopPageN], [printFileName As String])
```

Parameters

Parameter	Description
[promptUser]	Specifies Boolean value indicating if the user should be prompted for printer options.
[numberOfCopy]	Specifies the number of report copies you want printed.
[collated]	Specifies Boolean value specifying whether or not you want the report copies collated.
[startPageN]	Specifies the first page that you want printed.
[stopPageN]	Specifies the last page that you want printed.
[printFileName]	String. Specifies the name of the export print file.

ReadRecords Method (Report Object)

Use the ReadRecords method to force the report to read all records in the report from the database.

Syntax

```
Sub ReadRecords ()
```

SaveAs Method (Report Object)

Use the SaveAs method to save a report with the ability to specify Crystal Reports 8 or 7 version file format. The user can specify which format to use when saving the report. This call can only be made in formatting idle mode.

Syntax

```
Sub SaveAs (pFilePath As String, fileFormat As CRReportFileFormat)
```

Parameters

Parameter	Description
pFilePath	Specifies the file path and name that you want to use to save the report.
fileFormat	"CRReportFileFormat". Specifies the file format that you want to use to save the report.

SelectPrinter Method (Report Object)

The SelectPrinter method selects a different printer for the report. This method can be invoked only when in formatting idle mode.

Syntax

```
Sub SelectPrinter (pDriverName As String, pPrinterName As String, _  
pPortName As String)
```

Parameters

Parameter	Description
pDriverName	Specifies the name of the printer driver for the selected printer.
pPrinterName	Specifies the printer name for the selected printer.
pPortName	Specifies the port name for the port to which the selected printer is attached.

SetDialogParentWindow Method (Report Object)

The SetDialogParentWindow method sets the dialog parent window.

Syntax

```
Sub SetDialogParentWindow (hWnd As Long)
```

Parameter

Parameter	Description
hWnd	Specifies the handle of the parent window.

SetReportVariableValue Method (Report Object)

Use the SetReportVariableValue method to set the value of a report variable into the Print Engine. The report variable is designed to be used in Format Event to do calculated fields. However, you should not depend on how many times an event is fired to do total calculations—make sure that the calculation for the report variable is correct. The Print Engine keeps track of the status. This call can be made only in formatting active mode.

Syntax

```
Sub SetReportVariableValue (pName As String, var)
```

Parameters

Property	Description
pName	Specifies the name of the variable for which you want to set the value.
var	Specifies the value that you want to set.

SetUserPaperSize Method (Report Object)

Use the SetUserPaperSize method to set the paper length and width for the report.

Syntax

```
Sub SetUserPaperSize (length As Integer, Width As Integer)
```

Parameters

Parameter	Description
length	Integer. Specifies the length of the report in pixels.
Width	Integer. Specifies the width of the report in pixels.

Report Object Events

The following events are discussed in this section:

- “AfterFormatPage Event (Report Object)” on page 127
- “BeforeFormatPage Event (Report Object)” on page 128
- “FieldMapping Event (Report Object)” on page 128
- “NoData Event (Report Object)” on page 128

AfterFormatPage Event (Report Object)

The AfterFormatPage event is fired after formatting a page.

Syntax

```
Event AfterFormatPage (PageNumber As Long)
```

Parameter

Parameter	Description
PageNumber	Specifies the number of the report page triggering the event.

BeforeFormatPage Event (Report Object)

The BeforeFormatPage event is fired before formatting a page.

Syntax

```
Event BeforeFormatPage (PageNumber As Long)
```

Parameter

Parameter	Description
PageNumber	Specifies the number of the report page triggering the event.

FieldMapping Event (Report Object)

The FieldMapping event fires if the database is changed while verifying database.

Syntax

```
Event FieldMapping (reportFieldArray, databaseFieldArray, _  
    useDefault As Boolean)
```

Parameters

Parameter	Description
reportFieldArray	Specifies the report field array to map.
databaseFieldArray	Specifies the database field array to map.
useDefault	If TRUE, the values passed with parameters reportFieldArray and databaseFieldArray will be ignored and the default values used; if FALSE, the values passed with reportFieldArray and databaseFieldArray will be used.

NoData Event (Report Object)

The NoData event fires when there is no data for the report.

Syntax

```
Event NoData (pCancel As Boolean)
```

Parameter

Parameter	Description
pCancel	Specifies whether to cancel the report.

ReportAlert Object

The ReportAlert object represents a Report Alert contained in a report. This object provides properties for getting and setting information on Report Alerts in the report.

Report Alerts are custom messages created in either the Crystal Reports Designer, the Report Designer Component, or the Embeddable Crystal Reports Designer Control. Report Alerts may indicate action to be taken by the user or information about report data. For more information, see “Report Alerts” in the *Crystal Reports User’s Guide*.

ReportAlert Object Properties

Property	Definition	Read/Write	Restriction in event handler
AlertInstances	“ ReportAlertInstances Collection ”. Gets a reference to a collection of all the Report Alert instances created when the Report Alert is run.	Read only	None
Condition Formula	String. Gets or sets the condition formula for the Report Alert. The condition formula can be based on recurring records or on summary fields, but cannot be based on print-time fields, such as running totals or print time formulas. Condition formulas cannot have shared variables.	Read/Write	Can be written only when formatting idle.
Default Message	String. Gets or sets the default message for the Report Alert.	Read/Write	Can be written only when formatting idle.
IsEnabled	Boolean. Gets or sets whether or not the Report Alert is enabled.	Read/Write	Can be written only when formatting idle.
Message Formula	String. Gets or sets the message when the Report Alert is triggered. The result of the formula must be a string, and is created by combining a string with a report field. If MessageFormula is set it will override the value set for DefaultMessage.	Read/Write	Can be written only when formatting idle.
Name	String. Gets or sets the name of the Report Alert.	Read/Write	Can be written only when formatting idle.
Parent	“ Report Object ”. Gets reference to the parent object.	Read only	None

ReportAlerts Collection

The ReportAlerts collection contains the Report Alert objects defined in a report. Access a specific “[ReportAlert Object](#)” in the collection using the Item property.

ReportAlerts Collection Properties

Property	Definition	Read/Write	Restriction in event handler
Count	Long. Gets the number of “ ReportAlert Object ”, in the collection.	Read only	None
Item (indexAs Long)	“ ReportAlert Object ”. Gets an item from the Collection. Item has an index parameter that is a 1-based index (for example, Item(1) for the first Report Alert in the collection). The items in the collection are indexed in the order that they were added to the report.	Read only	None
Parent	“ Report Object ”. Gets reference to the parent object.	Read only	None

ReportAlerts Collection Methods

The following methods are discussed in this section:

- “[Add Method \(ReportAlerts Collection\)](#)” on page 130
- “[Delete Method \(ReportAlerts Collection\)](#)” on page 131

Add Method (ReportAlerts Collection)

The Add method is used to add a Report Alert object to the report.

Syntax

```
Function Add (Name As String, DefaultMessage As String, IsEnabled As Boolean, ConditionFormula As String, [MessageFormula as String]) As ReportAlert
```

Parameter	Description
Name	Specifies the name of the Report Alert.
DefaultMessage	Specifies the default message created by the Report Alert.
IsEnabled	Specifies whether or not the Report Alert is enabled when the report is run.
ConditionFormula	Specifies the conditional formula that evaluates when the Report Alert is triggered.
MessageFormula	Optional. Formula used to create the message when the Report Alert is triggered. The result of the formula must be a string, and is created by combining a string with a report field. If MessageFormula is set it will override the value set for DefaultMessage.

Sample

In this example a report is grouped by country and contains a summary of last year's sales per country. The Report Alert will be triggered when the summary of last year's sales exceeds \$25,000.00. When the Report Alert is triggered a message is created from the default message and the country name.

```
Dim name, defaultMessage, conditionFormula, messageFormula As String

'Name of the Report Alert
name = "Sales Alert"

defaultMessage = "Great sales in "

' Conditional formula used to evaluate the Report Alert.
conditionFormula = "Sum ({Customer.Last Year's Sales}, {Customer.Country}) > 25000"

' Replaces the default message set for the Report Alert. The message is
' created in a formula by concatenating the value set in the
' DefaultMessage parameter, and the group name of the country that
' triggers the Report Alert.
' DefaultAttribute is a function used by the Formula Editor to return
' the default message set for the Report Alert.
messageFormula = "DefaultAttribute & GroupName ({Customer.Country})"

Report.ReportAlerts.Add _
name, defaultMessage, True, conditionFormula, messageFormula
```

Delete Method (ReportAlerts Collection)

Use the Delete method to remove a ReportAlert Object from the Collection.

Syntax

```
Sub Delete (index)
```

Parameter

Parameter	Description
index	Specifies the index number of the Object that you want to remove from the Collection.

ReportAlertInstance Object

A ReportAlertInstance object is created each time a Report Alert is triggered. This object contains a property for getting the message created for that specific instance of the Report Alert.

Note: In the present build only the first ReportAlertInstance is created at runtime. This limitation will be addressed in a future release.

ReportAlertInstance Object Properties

Property	Definition	Read/Write	Restriction in event handler
AlertMessage	String. Gets the message returned at the time the Report Alert is triggered.	Read only	None
Parent	"ReportAlert Object". Gets reference to the parent object.	Read only	None

ReportAlertInstances Collection

The ReportAlertInstances collection contains the ReportAlertInstance objects created when a Report Alert is triggered. Access a specific "ReportAlertInstance Object" in the collection using the Item property.

Note: In the present build only the first ReportAlertInstance is created at runtime. This limitation will be addressed in a future release.

ReportAlertInstances Properties

Property	Definition	Read/Write	Restriction in event handler
Count	Long. Gets the number of "ReportAlertInstance Object" in the collection.	Read only	None
Item (indexAs Long)	"ReportAlertInstance Object". Gets an item from the Collection. Item has an index parameter that is a 1-based index (for example, Item(1) for the first ReportAlertInstance in the collection). The items in the collection are indexed in the order that they were added to the report. Note: In the present build only the first ReportAlertInstance is created at runtime. This limitation will be addressed in a future release.	Read only	None
Parent	"ReportAlert Object". Gets reference to the parent object.	Read only	None

ReportObjects Collection

The ReportObjects Collection is a collection of report objects in a section. Report objects can be a field, text, OLE, cross-tab, subreport, BLOB field, Box, Graph, Line, Map, or OlapGrid objects. Access a specific object in the collection using the Item property.

ReportObjects Collection Properties

Property	Description	Read/Write	Restriction in event handler
Count	Long. Gets the number of report objects in the collection.	Read only	None
Item (index) As Object	Gets a report object. Depending on the type of item referenced, this can be a "BlobFieldObject Object", "FieldMappingData Object", "TextObject Object", "Page Object", "CrossTabObject Object", "BoxObject Object", "Remarks", "GraphObject Object", "LineObject Object", "MapObject Object", "OlapGridObject Object". Item has an index parameter that is a numeric, 1-based index (that is, Item (1)). The items in the collection are indexed in the order they were added to the report.	Read only	None
Parent	"Section Object". Gets reference to the parent object.	Read only	None

Remarks

Item is a default property. You can reference a report object directly, for example, ReportObjects(1).

RunningTotalFieldDefinition Object

The RunningTotalFieldDefinition Object represents a running total field used in the report. This object provides properties for getting information on running total fields in the report.

RunningTotalFieldDefinition Object Properties

Property	Definition	Read/Write	Restriction in event handler
EvaluateCondition	"CRRunningTotalCondition". Gets evaluate condition.	Read only	None
EvaluateCondition Field	Object. Gets evaluate condition field.	Read only	None

Property	Definition	Read/Write	Restriction in event handler
EvaluateCondition Formula	String. Gets or sets evaluate condition formula.	Read/Write	Can be written only when formatting idle.
EvaluateGroup Number	Integer. Gets or sets evaluate group number.	Read/Write	Can be written only when formatting idle.
Hierarchical SummaryType	<i>“CRHierarchicalSummaryType”</i> . Gets or sets whether or not to calculate the running total across the hierarchy in a hierarchically grouped report.	Read/Write	Can be written only when formatting idle.
Kind	<i>“CRFieldKind”</i> . Gets which kind of field (that is, database, summary, formula, etc.).	Read only	None
Name	String. Gets the unique formula name of the field within the report (table.FIELD). For example, {#Test}.	Read only	None
NextValue	Variant. Gets the field next value.	Read only	Can be read only when top-level Report object is formatting active.
NumberOfBytes	Integer. Gets the number of bytes required to store the field data in memory.	Read only	None
Parent	<i>“Report Object”</i> . Gets reference to the parent object.	Read only	None
PreviousValue	Variant. Gets the field previous value.	Read only	Can be read only when top-level Report object is formatting active.
ResetCondition	<i>“CRRunningTotalCondition”</i> . Gets the reset condition.	Read only	None
ResetConditionField	Object. Gets reset condition field.	Read only	None
ResetCondition Formula	String. Gets or sets the reset condition formula.	Read/Write	Can be written only when formatting idle.
ResetGroupNumber	Integer. Gets or sets the reset group number.	Read/Write	Can be written only when formatting idle.
RunningTotalField Name	String. Gets the running total field name.	Read only	None
Secondary SummarizedField	Object. Gets the secondary summarized field.	Read only	Can be written only when formatting idle.
SummarizedField	Object. Gets the summarized field.	Read only	Can be written only when formatting idle.

Property	Definition	Read/Write	Restriction in event handler
SummaryOperationParameter	Long. Gets or sets summary operation parameter.	Read/Write	Can be written only when formatting idle.
SummaryType	"CRSummaryType". Gets or sets summary type.	Read/Write	Can be written only when formatting idle.
Value	Variant. Gets the field current value.	Read only	Can be read only when top-level Report object is formatting active.
ValueType	"CRFieldValueType". Gets which type of value is found in the field.	Read only	None

RunningTotalFieldDefinition Object Methods

The following methods are discussed in this section:

- "SetEvaluateConditionField Method (RunningTotalFieldDefinition Object)" on page 135
- "SetNoEvaluateCondition Method (RunningTotalFieldDefinition Object)" on page 136
- "SetNoResetCondition Method (RunningTotalFieldDefinition Object)" on page 136
- "SetResetConditionField Method (RunningTotalFieldDefinition Object)" on page 136
- "SetSecondarySummarizedField Method (RunningTotalFieldDefinition Object)" on page 136
- "SetSummarizedField Method (RunningTotalFieldDefinition Object)" on page 137

SetEvaluateConditionField Method (RunningTotalFieldDefinition Object)

Use the SetEvaluateConditionField to set the evaluate condition field.

Syntax

```
Sub SetEvaluateConditionField (pEvaluateConditionField)
```

Parameter

Parameter	Description
pEvaluateConditionField	Specifies the condition field that you want to use.

SetNoEvaluateCondition Method (RunningTotalFieldDefinition Object)

Use the SetNoEvaluateCondition method to specify no evaluate condition for the RunningTotalFieldDefinition Object.

Syntax

```
Sub SetNoEvaluateCondition ()
```

SetNoResetCondition Method (RunningTotalFieldDefinition Object)

Use the SetNoResetCondition method to specify no reset condition for the RunningTotalFieldDefinition Object.

Syntax

```
Sub SetNoResetCondition ()
```

SetResetConditionField Method (RunningTotalFieldDefinition Object)

Use the SetResetConditionField to specify the reset condition field to use with the RunningTotalFieldDefinition Object.

Syntax

```
Sub SetResetConditionField (pResetConditionField)
```

Parameter

Parameter	Description
pResetConditionField	Specifies the condition field that you want to use.

SetSecondarySummarizedField Method (RunningTotalFieldDefinition Object)

Use the SetSecondarySummarizedField method to specify which condition field you want to use as the second summarized field for the RunningTotalFieldDefinition Object.

Syntax

```
Sub SetSecondarySummarizedField (secondarySummariedField)
```

Parameter

Parameter	Description
SecondarySummarizedField	Specifies the condition field that you want to use.

SetSummarizedField Method (RunningTotalFieldDefinition Object)

Use the SetSummarizedField method to specify a second summarized field for the RunningTotalFieldDefinition Object.

Syntax

Sub SetSummarizedField (SummarizedField)

Parameter

Parameter	Description
SummarizedField	Specifies the condition field that you want to use.

RunningTotalFieldDefinitions Collection

The RunningTotalFieldDefinitions Collection is a collection of running total field definition objects. One object exists in the collection for every running total field accessed by the report. Access a specific *“RunningTotalFieldDefinition Object”* in the collection using the Item property.

RunningTotalFieldDefinitions Collection Properties

Property	Description	Read/Write	Restriction in event handler
Count	Long. Gets the number of <i>“RunningTotalFieldDefinition Object”</i> in the collection.	Read only	None
Item (indexAs Long)	<i>“RunningTotalFieldDefinition Object”</i> . Gets an item from the Collection. Item has an index parameter that is a 1-based index (for example, Item(1) for the first database field in the collection). The items in the collection are indexed in the order that they were added to the report.	Read only	None
Parent	<i>“Report Object”</i> . Gets reference to the parent object.	Read only	None

Remarks

Instead of using the Item property as shown, you can reference a database directly, for example, RunningTotalFieldDefinitions(1).

RunningTotalFieldDefinitions Collection Methods

The following methods are discussed in this section:

- *“Add Method (RunningTotalFieldDefinitions Collection)”* on page 138
- *“Delete Method (RunningTotalFieldDefinitions Collection)”* on page 138
- *“GetItemByName Method (RunningTotalFieldDefinitions Collection)”* on page 138

Add Method (RunningTotalFieldDefinitions Collection)

Use the Add method to add a “RunningTotalFieldDefinition Object” to the Collection.

Syntax

```
Function Add (runningTotalName As String) As RunningTotalFieldDefinition
```

Parameter

Parameter	Description
runningTotalName	Specifies the name of the RunningTotal field that you want to add.

Returns

Returns a “RunningTotalFieldDefinition Object” member of the Collection.

Delete Method (RunningTotalFieldDefinitions Collection)

Use the Delete method to remove a RunningTotalFieldDefinition Object from the Collection.

Syntax

```
Sub Delete (index)
```

Parameter

Parameter	Description
index	Specifies the index number of the Object that you want to remove from the Collection.

GetItemByName Method (RunningTotalFieldDefinitions Collection)

Use the GetItemByName method to retrieve a RunningTotalFieldDefinition object by name.

Syntax

```
Sub GetItemByName(name As String, runningTotalFieldDefinition)
```

Parameters

Parameter	Description
name	The item’s unique name.
runningTotalFieldDefinition	A RunningTotalFieldDefinition object to hold the retrieved item.

Section Object

Report areas contain at least one section. The Section Object includes properties for accessing information regarding a section of your report. This object holds on to a report object, then releases the report object when it is destroyed.

Section Object Properties

Property	Description	Read/Write	Restriction in event handler
BackColor	OLE_COLOR. Gets or sets the section background color.	Read/Write	Can be written only when formatting idle or active.
Condition Formula	String. Gets or sets the condition formula. The condition formula can be based on recurring records or on summary fields, but it cannot be based on print-time fields, such as running totals or print-time formulas. Condition formulas cannot have shared variables.	Read/Write	Can be written only when formatting idle.
CssClass	String. Gets or sets the cascading style sheet Class.	Read/Write	Can be written only when formatting idle.
Height	Long. Gets or sets the height of the section, in twips.	Read/Write	Can be written only when formatting idle or active.
KeepTogether	Boolean. Gets or sets the option that indicates whether to keep the entire section on the same page if it is split into two pages.	Read/Write	Can be written only when formatting idle or active.
MinimumHeight	Long. Gets the minimum section height, in twips.	Read only	None
Name	String. Gets or sets the name of the section.	Read/Write	Can be written only when formatting idle.
NewPageAfter	Boolean. Gets or sets the option that indicates whether to start a new page after the current section.	Read/Write	Can be written only when formatting idle or active.
NewPageBefore	Boolean. Gets or sets the option that indicates whether to start a new page before the current section.	Read/Write	Can be written only when formatting idle or active.
Number	Integer. Gets the index number associated with the section in the area (for example, if the first section in an area, the number returned is 1).	Read only	None

Property	Description	Read/Write	Restriction in event handler
Parent	"Area Object". Gets reference to the parent object.	Read only	None
PrintAtBottomOfPage	Boolean. Gets or sets the option that indicates whether to print the current section at the bottom of the page.	Read/Write	Can be written only when formatting idle or active.
ReportObjects	"ReportObjects Collection". Gets reference to the heterogeneous collection of report objects for the section.	Read only	None
ResetPageNumberAfter	Boolean. Gets or sets the option that indicates whether to reset the page number after the current section.	Read/Write	Can be written only when formatting idle or active.
Suppress	Boolean. Gets or sets the section visibility.	Read/Write	Can be written only when formatting idle or active.
SuppressIfBlank	Boolean. Gets or sets the option that indicates whether to suppress the current section if it is blank.	Read/Write	Can be written only when formatting idle or active.
UnderlaySection	Boolean. Gets or sets the underlay following section option.	Read/Write	Can be written only when formatting idle or active.
Width	Long. Gets the width of the section, in twips.	Read only	None

Section Object Methods

The following methods are discussed in this section:

- "AddBlobFieldObject Method (Section Object)" on page 141
- "AddBoxObject Method (Section Object)" on page 141
- "AddCrossTabObject Method (Section Object)" on page 142
- "AddFieldObject Method (Section Object)" on page 142
- "AddGraphObject Method (Section Object)" on page 143
- "AddLineObject Method (Section Object)" on page 143
- "AddPictureObject Method (Section Object)" on page 144
- "AddSpecialVarFieldObject Method (Section Object)" on page 144
- "AddSubreportObject Method (Section Object)" on page 145
- "AddSummaryFieldObject Method (Section Object)" on page 145
- "AddTextObject Method (Section Object)" on page 146
- "AddUnboundFieldObject Method (Section Object)" on page 147
- "DeleteObject Method (Section Object)" on page 147
- "ImportSubreport Method (Section Object)" on page 147

AddBlobFieldObject Method (Section Object)

The AddBlobFieldObject method adds a “BlobFieldObject Object” to the Section Object.

Syntax

```
Function AddBlobFieldObject (Field, Left As Long, Top As Long _  
    ) As BlobFieldObject
```

Parameters

Parameter	Description
Field	Variant. Can be formula form name or any field definition that specifies the field that you want to add.
Left	Specifies the offset of the top left corner of the Object that you are adding relative to the top left corner of the parent Section, in twips.
Top	Specifies the offset of the top left corner of the Object that you are adding relative to the top left corner of the parent Section, in twips.

Returns

Returns a “BlobFieldObject Object”.

AddBoxObject Method (Section Object)

The BoxObject method adds a “BoxObject Object” to the Section Object.

Syntax

```
Function AddBoxObject (Left As Long, Top As Long, Right As Long, _  
    Bottom As Long, [pEndSection]) As BoxObject
```

Parameters

Parameter	Description
Left	Specifies the offset of the top left corner of the Object that you are adding relative to the top left corner of the parent Section, in twips.
Top	Specifies the offset of the top left corner of the Object that you are adding relative to the top left corner of the parent Section, in twips.
Right	Specifies the offset of the bottom right corner of the Object that you are adding relative to the bottom right corner of the parent (or end) Section, in twips.
Bottom	Specifies the offset of the bottom right corner of the Object that you are adding relative to the bottom right corner of the parent (or end) Section, in twips.
[pEndSection]	Specifies the Section in which the end of the BoxObject will be placed, if not the parent Section.

Returns

Returns a “BoxObject Object”.

AddCrossTabObject Method (Section Object)

The AddCrossTabObject method adds a “CrossTabObject Object” to the Section Object. This method creates an empty CrossTabObject. Use the CrossTabObject properties and methods to add grouping and summary info.

Syntax

```
Function AddCrossTabObject (Left As Long, Top As Long) As CrossTabObject
```

Parameters

Parameter	Description
Left	Specifies the offset of the top left corner of the Object that you are adding relative to the top left corner of the parent Section, in twips.
Top	Specifies the offset of the top left corner of the Object that you are adding relative to the top left corner of the parent Section, in twips.

Returns

Returns a “CrossTabObject Object”.

AddFieldObject Method (Section Object)

The AddFieldObject method adds a “FieldMappingData Object” to the Section Object. The FieldObject can be reference to a database field definition, formula field definition, running total field definition, SQL expression field definition, or parameter field definition.

Syntax

```
Function AddFieldObject (Field, Left As Long, Top As Long) As FieldObject
```

Parameters

Parameter	Description
Field	Specifies the field that you want to add to the Section.
Left	Specifies the offset of the top left corner of the Object that you are adding relative to the top left corner of the parent Section, in twips.
Top	Specifies the offset of the top left corner of the Object that you are adding relative to the top left corner of the parent Section, in twips.

Returns

Returns a “FieldMappingData Object”.

AddGraphObject Method (Section Object)

The AddGraphObject method adds a “**GraphObject Object**” (Chart) to the Section Object. The inserted GraphObject is empty. Use the GraphObject properties and methods to add data, groups and settings. If optional parameter [pCrossTabObject] is passed, you can insert a CrossTab Chart.

Syntax

```
Function AddGraphObject (graphDataType As CRGraphDataType, _
    Left As Long, Top As Long, [pCrossTabObject]) As GraphObject
```

Parameters

Parameter	Description
graphDataType	“ CRGraphDataType ”. Specifies the data type for the graph that you want to add.
Left	Specifies the offset of the top left corner of the Object that you are adding relative to the top left corner of the parent Section, in twips.
Top	Specifies the offset of the top left corner of the Object that you are adding relative to the top left corner of the parent Section, in twips.
[pCrossTabObject]	If graphDataType=crCrossTabGraph, this parameter specifies the CrossTabObject to associate with the chart.

Returns

Returns a “**GraphObject Object**”.

AddLineObject Method (Section Object)

The AddLineObject method adds a “**LineObject Object**” to the Section Object. If optional parameter [pEndSection] is passed different from the current section, you can add a vertical line across sections.

Syntax

```
Function AddLineObject (Left As Long, Top As Long, _
    Right As Long, Bottom As Long, [pEndSection]) As LineObject
```

Parameters

Parameter	Description
Left	Specifies the offset of the top left corner of the Object that you are adding relative to the top left corner of the parent Section, in twips.
Top	Specifies the offset of the top left corner of the Object that you are adding relative to the top left corner of the parent Section, in twips.

Parameter	Description
Right	Specifies the offset of the bottom right corner of theObject that you are adding relative to the bottom right corner of the parent (or end) Section, in twips.
Bottom	Specifies the offset of the bottom right corner of theObject that you are adding relative to the bottom right corner of the parent (or end) Section, in twips.
[pEndSection]	Specifies the Section in which the end of the LineObject will be placed, if not the parent Section.

Returns

Returns a “LineObject Object”.

AddPictureObject Method (Section Object)

The AddLineObject method adds a picture object from an image in the form of an “OleObject Object” to the Section Object.

Syntax

```
Function AddPictureObject (pImageFilePath As String, Left As Long, _
    Top As Long) As OleObject
```

Parameters

Parameter	Description
pImageFilePath	Specifies the image file path and name that you want to add.
Left	Specifies the offset of the top left corner of the Object that you are adding relative to the top left corner of the parent Section, in twips.
Top	Specifies the offset of the top left corner of the Object that you are adding relative to the top left corner of the parent Section, in twips.

Returns

Returns an “OleObject Object”.

AddSpecialVarFieldObject Method (Section Object)

The SpecialVarFieldObject Method adds a SpecialVar “FieldMappingData Object” to the Section Object.

Syntax

```
Function AddSpecialVarFieldObject (specialVarType As CRSpecialVarType, _
    Left As Long, Top As Long) As FieldObject
```

Parameters

Parameter	Description
specialVarType	"CRSpecialVarType". Specifies the SpecialVar type.
Left	Specifies the offset of the top left corner of the Object that you are adding relative to the top left corner of the parent Section, in twips.
Top	Specifies the offset of the top left corner of the Object that you are adding relative to the top left corner of the parent Section, in twips.

Returns

Returns a "FieldMappingData Object".

AddSubreportObject Method (Section Object)

The SubReportObject method adds a "SubreportObject Object" to the Section Object. This method adds an empty subreport to the Section. You can then add Objects and Sections to the SubreportObject.

Syntax

```
Function AddSubreportObject (pSubreportName As String, _
    Left As Long, Top As Long) As SubreportObject
```

Parameters

Parameter	Description
pSubreportName	Specifies the name of the subreport that you want to add.
Left	Specifies the offset of the top left corner of the Object that you are adding relative to the top left corner of the parent Section, in twips.
Top	Specifies the offset of the top left corner of the Object that you are adding relative to the top left corner of the parent Section, in twips.

Returns

Returns a "SubreportObject Object".

AddSummaryFieldObject Method (Section Object)

The AddSummaryField method adds a summary "FieldMappingData Object" to the Section Object.

Syntax

```
Function AddSummaryFieldObject (Field, SummaryType As CRSummaryType, _
    Left As Long, Top As Long, [secondSummaryFieldOrFactor]) As FieldObject
```

Parameters

Parameter	Description
Field	Specifies the summary field that you want to add.
SummaryType	“CRSummaryType” . Specifies the type of summary field.
Left	Specifies the offset of the top left corner of the Object that you are adding relative to the top left corner of the parent Section, in twips.
Top	Specifies the offset of the top left corner of the Object that you are adding relative to the top left corner of the parent Section, in twips.
[secondSummaryFieldOrFactor]	Specifies the second summary field or factor, if required by the summary type.

Returns

Returns a **“FieldMappingData Object”**.

AddTextObject Method (Section Object)

The AddTextObject method adds a **“TextObject Object”** to the Section Object. String parameter pText can contain formatting information such as tab stops and carriage returns.

Syntax

```
Function AddTextObject (pText As String, Left As Long, Top As Long, _  
[formatText]) As TextObject
```

Parameters

Parameter	Description
pText	Specifies the text that you want to add.
Left	Specifies the offset of the top left corner of the Object that you are adding relative to the top left corner of the parent Section, in twips.
Top	Specifies the offset of the top left corner of the Object that you are adding relative to the top left corner of the parent Section, in twips.
[formatText]	Boolean. If TRUE, the text will be formatted according to the formatting characters included in the text string (for example, carriage returns, tabs). If FALSE, formatting characters in the text string will be ignored.

Returns

Returns a **“FieldMappingData Object”**.

AddUnboundFieldObject Method (Section Object)

The AddUnboundFieldObject method adds an unbound “FieldMappingData Object” to the Section Object. The unbound field can be bound to a Crystal Report formula or a data source at runtime.

Syntax

```
Function AddUnboundFieldObject (ValueType As CRFieldValueType, _  
    Left As Long, Top As Long) As FieldObject
```

Parameters

Parameter	Description
ValueType	“CRFieldValueType”. Specifies the type of value in the field.
Left	Specifies the offset of the top left corner of the Object that you are adding relative to the top left corner of the parent Section, in twips.
Top	Specifies the offset of the top left corner of the Object that you are adding relative to the top left corner of the parent Section, in twips.

Returns

Returns a “FieldMappingData Object”.

DeleteObject Method (Section Object)

The DeleteObject method removes an object from the Section Object.

Syntax

```
Sub DeleteObject (reportObject)
```

Parameter

Parameter	Description
reportObject	Specifies the report object that you want to remove from the Section.

ImportSubreport Method (Section Object)

Examples

“How to import a subreport and link to a main report” on page 193

The ImportSubreport method imports a “SubreportObject Object” into the Section Object. With this call you can insert a subreport object obtained from an existing report file. The inserted Subreport Object will have all of the Objects and Sections of the original (source) report, except for Page Header and Page Footer Sections.

Syntax

```
Function ImportSubreport (subreportFileName As String, _
    Left As Long, Top As Long) As SubreportObject
```

Parameters

Parameter	Description
subreportFileName	Specifies the name of the subreport that you want to import.
Left	Specifies the offset of the top left corner of the Object that you are adding relative to the top left corner of the parent Section, in twips.
Top	Specifies the offset of the top left corner of the Object that you are adding relative to the top left corner of the parent Section, in twips.

Returns

Returns a "SubreportObject Object".

Section Object Events

The following event is discussed in this section:

- ["Format Event \(Section Object\)" on page 148](#)

Format Event (Section Object)**Examples**

["How to format a group in the section format event" on page 200](#)

The Format event is fired before the print engine starts formatting a section. In the Format event handler you can use the object model to write some code to change the outcome of the formatted section. Be aware, however, that not all of the properties and methods provided in the object model are accessible at all times in the event handler. Specifically, only those properties and methods that have been marked "formatting active" or "no restrictions" can be used in the Event.

Syntax

```
Event Format (FormattingInfo As Object)
```

Parameter

Parameter	Description
FormattingInfo	The FormattingInfo object which will contain formatting information to be used by the Format event handler.

Remarks

The following comments regarding Restrictions and Rules apply to Format Event:

Restrictions

The Format event handler should not have any state. No matter when or how many times it is called it should always behave the same way. It should not keep track of how many times it is called and then, for example, change the background color based on how many times it has been called. The print engine formatting procedure is very complicated and one section can be formatted many times depending on different situations.

Note: Because the Format event for each section may be fired more than once during page printing you should not use it to do any totaling in which values are carried over from one section to another. However, you can use report variables to do calculated fields.

There are three formatting modes:

FormattingIdle	No formatting occurs. This can be before going to preview or print or between changing pages in the viewer.
FormattingActive	The start of formatting an object. The object mode is marked as formatting active.
FormattingInactive	When the print engine is formatting and one object is in FormattingActive mode the rest of the objects are in FormattingInactive mode.

Since there is only one section Format event when one section is formatting all of the objects in that section are in FormattingActive mode and all the rest of the sections and objects are in FormattingInactive mode.

Rules

- The read property can be accessed in any formatting mode. (There are a few exceptions, please see the object model reference section.)
- Top level Report, Areas and Area write properties and methods can only be accessed in formatting idle mode, aside from a few exceptions.
- When in FormattingIdle mode you should make changes before going to preview or print. Making changes between changing pages may result in an unexpected result.
- For sections and objects within sections write property or method access is denied when the section is in FormattingInactive mode.
- For sections and objects within sections some write property or method access is permitted and some denied while the section is in FormattingActive mode. Please refer the object model reference for details.

Sections Collection

Sections can come from either the “**Report Object**” or “**Area Object**”. The Sections Collection is a collection of section objects. Access a specific Section Object in the collection using the Item property.

- When from the Report Object, the sections object will contain all the sections in the report.
- When from the Area Object, the sections object will contain all the sections in the area only.

Sections Collection Properties

Property	Description	Read/Write	Restriction in event handler
Count	Long. Gets the number of section in the collection.	Read only	None
Item (index)	“ Section Object ”. Gets reference to an item in the Collection. Item has an index parameter that can be either a string reference to the area section (i.e., for areas with one section: “RH”, “PH”, “GHn”, “D”, “GFn”, “PF”, or “RF”) or a numeric, 1-based index (i.e., Item (1)) for the Report Header area. Numeric index for sections starts at 1 for first section in the area/report and continues in order of appearance. If the area has multiple sections, they are represented using a lowercase letter (for example, “Da”, “Db).	Read only	None
Parent	“ Report Object ” or “ Area Object ”. Gets reference to the parent object.	Read only	None

Remarks

Instead of using the Item property as shown, you can reference a section directly, for example, Sections (“Da”).

Sections Collection Methods

The following methods are discussed in this section:

- “**Add Method (Sections Collection)**” on page 150
- “**Delete Method (Sections Collection)**” on page 151

Add Method (Sections Collection)

Use Add method to add a “**Section Object**” to the Sections Collection.

Syntax

Function Add ([index]) As Section

Parameter

Parameter	Description
[index]	Specifies the index where you would like to add the Section to the Collection.

Returns

Returns a “Section Object” member of the Sections Collection.

Delete Method (Sections Collection)

Use Delete method to remove a “Section Object” from the Sections Collection.

Syntax

```
Sub Delete(index)
```

Parameter

Parameter	Description
index	Specifies the index of the Section that you want to delete from the Collection.

SortField Object

The SortField Object includes properties for accessing information for record or group sort fields. It holds on to Report object, then releases the report object when destroyed. This object has an index instance variable to indicate its index.

SortField Object Properties

Property	Description	Read/Write	Restriction in event handler
Field	Object. Gets or sets the sorting field definition object.	Read/Write	Can be read or written only when formatting idle.
Parent	“Report Object”. Gets reference to the parent object.	Read only	None
SortDirection	“CRSortDirection”. Gets or sets the sort direction.	Read/Write	Can be written only when formatting idle.

SortFields Collection

The SortFields Collection is a collection of sort fields; can be either record sort field or group sort field. Access a specific SortField Object in the collection using the Item property.

SortFields Collection Properties

Property	Description	Read/Write	Restriction in event handler
Count	Long. Gets the number of sort fields in the collection.	Read only	None
Item (index As Long)	"SortField Object". Gets reference to an item in the Collection. Item has an index parameter that is a numeric, 1-based index (that is, Item (1)). The sort fields in the collection are indexed in the order they were added as sort fields to the report.	Read only	None
Parent	"Report Object". Gets reference to the parent object.	Read only	None

Remarks

Instead of using the Item property as shown, you can reference a sort field directly, for example, SortFields(1).

SortFields Collection Methods

The following methods are discussed in this section:

- "Add Method (SortFields Collection)" on page 152
- "Delete Method (SortFields Collection)" on page 153

Add Method (SortFields Collection)

The Add method adds a record or group sort field to the Collection.

Syntax

```
Sub Add (pFieldDefinition As IFieldDefinition, _
        SortDirection As CRSortDirection)
```

Parameters

Parameter	Description
pFieldDefinition	A field definition object. Specifies the field definition or field name.
SortDirection	"CRSortDirection". Specifies the direction in which the field data should be sorted (that is, ascending, descending, etc.).

Delete Method (SortFields Collection)

The Delete method deletes a record or group sort field from the Collection.

Syntax

Sub Delete (index As Long)

Parameter

Parameter	Description
index	Specifies the 1-based index number of the sort field in the collection that you want to delete.

SpecialVarFieldDefinition Object

The SpecialVarFieldDefinition Object provides properties for retrieving information and setting options for a special field found in your report (for example, last modification date, print date). A SpecialVarFieldDefinition Object is obtained from the Field property of the *FieldMappingData Object*.

SpecialVarFieldDefinition Object Properties

Property	Description	Read/Write	Restriction in event handler
Kind	<i>CRFieldKind</i> . Gets which kind of field (database, summary, formula, etc.).	Read only	None
Name	String. Gets the field definition unique formula name of the special var. field.	Read only	None
NextValue	Variant. Gets the field next value.	Read only	Can be read only when top-level Report object is formatting active.
NumberOfBytes	Integer. Gets the number of bytes required to store the field data in memory.	Read only	None
Parent	<i>Report Object</i> . Gets reference to the parent object.	Read only	None
PreviousValue	Variant. Gets the field previous value.	Read only	Can be read only when top-level Report object is formatting active.
SpecialVarType	<i>CRSpecialVarType</i> . Gets what the type of special field (for example, ReportTitle, PageNumber).	Read only	None

Property	Description	Read/Write	Restriction in event handler
Value	Variant. Gets the field current value.	Read only	Can be read only when top-level Report object is formatting active.
ValueType	" CRFieldValueType ". Gets which type of value is found in the field.	Read only	None

SQLExpressionFieldDefinition Object

The SQLExpressionFieldDefinition Object represents a SQL expression field used in the report. This object provides properties for getting information on SQL expression fields in the report.

SQLExpressionFieldDefinition Object Properties

Property	Definition	Read/Write	Restriction in event handler
Kind	" CRFieldKind ". Gets which kind of field (that is, database, summary, formula, etc.).	Read only	None
Name	String. Gets the field definition unique formula name of the field within the report, Crystal formula syntax.	Read only	None
NextValue	Variant. Gets the field next value.	Read only	Can be read only when top-level Report object is formatting active.
NumberOfBytes	Integer. Gets the number of bytes required to store the field data in memory.	Read only	None
Parent	" Report Object ". Gets reference to the parent object.	Read only	None
PreviousValue	Variant. Gets the field previous value.	Read only	Can be read only when top-level Report object is formatting active.
SQLExpressionFieldName	String. Gets the SQL expression field name.	Read only	None
Text	String. Gets or sets the SQL expression text.	Read/Write	Can be written only when formatting idle.
Value	Variant. Gets the field current value.	Read only	Can be read only when top-level Report object is formatting active.
ValueType	" CRFieldValueType ". Gets which type of value is found in the field.	Read only	None

SQLExpressionFieldDefinition Object Methods

The following method is discussed in this section:

- “Check Method (SQLExpressionFieldDefinition Object)” on page 155

Check Method (SQLExpressionFieldDefinition Object)

Use Check method to check whether a SQL expression is valid.

Syntax

```
Sub Check (pBool As Boolean, ppErrorString As String)
```

Parameters

Parameter	Description
pBool	Boolean value indicating the condition of the formula string. Will be set to TRUE if the formula is valid and FALSE if the formula contains one or more errors.
ppErrorString	Specifies the error message string if the formula contains an error.

SQLExpressionFieldDefinitions Collection

The SQLExpressionFieldDefinitions Collection is a collection of SQL expression field definition objects. One object exists in the collection for every SQL expression field accessed by the report. Access a specific “SQLExpressionFieldDefinition Object” in the collection using the Item property.

SQLExpressionFieldDefinitions Collection Properties

Property	Description	Read/Write	Restriction in event handler
Count	Long. Gets the number of items in the Collection.	Read only	None
Item (index As Long)	“SQLExpressionFieldDefinition Object”. Gets an item from the Collection. Item has an index parameter that is a 1-based index (that is, Item (1) for the first SQL Expression field in the collection). The items in the collection are indexed in the order they were added to the report.	Read only	None
Parent	“Report Object”. Gets reference to the parent object.	Read only	None

Remarks

Instead of using the Item property as shown, you can reference a database directly, for example, SQLExpressionFieldDefinitions(1).

SQLExpressionFieldDefinitions Collection Methods

The following methods are discussed in this section:

- “Add Method (SQLExpressionFieldDefinitions Collection)” on page 156
- “Delete Method (SQLExpressionFieldDefinitions Collection)” on page 156
- “GetItemByName Method (SQLExpressionFieldDefinitions Collection)” on page 157

Add Method (SQLExpressionFieldDefinitions Collection)

Use Add method to add a “SQLExpressionFieldDefinition Object” to the Collection.

Syntax

```
Function Add (SQLExpressionName As String, _
            Text As String) As SQLExpressionFieldDefinition
```

Parameters

Parameter	Description
SQLExpressionName	Specifies the name of the SQL expression that you want to add to the Collection.
Text	Specifies the text of the SQL expression that you want to add to the Collection.

Returns

Returns a “SQLExpressionFieldDefinition Object” member of the Collection.

Delete Method (SQLExpressionFieldDefinitions Collection)

Use Delete method to remove a “SQLExpressionFieldDefinition Object” from the Collection.

Syntax

```
Sub Delete (index)
```

Parameter

Parameter	Description
index	Specifies the index number of the SQLExpressionFieldDefinition Object that you want to remove from the Collection.

GetItemByName Method (SQLExpressionFieldDefinitions Collection)

Use GetItemByName method to retrieve an SQLExpressionFieldDefinition object by name.

Syntax

```
Sub GetItemByName(name As String, sqlExpressionFieldDefinition)
```

Parameters

Parameter	Description
name	The item's unique name.
sqlExpressionFieldDefinition	An SQLExpressionFieldDefinition object to hold the retrieved item.

SubreportLink Object

The SubreportLink Object provides information about Subreport Links to the main report.

SubreportLink Object Properties

Property	Description	Read/Write	Restriction in event handler
MainReportField	A field definition object. Gets the main report field to which the subreport field is linked. See Remarks .	Read only	None
Parent	"SubreportObject Object". Gets the subreport link's parent object.	Read only	None
SubReportField	A field definition object. Gets the subreport field. See Remarks .	Read only	None

Remarks

After getting the field definition object from the MainReportField or the SubReportField property, you can use the Kind property to determine what type of field definition object it is. All field definition objects have the Kind property.

SubreportLinks Collection

The SubreportLinks Collection contains the “SubreportLink Object” related to the report.

SubreportLinks Collection Properties

Property	Description	Read/Write	Restriction in event handler
Count	Long. Gets a count of the items in the Collection.	Read only	None
Item (index As Long)	“SubreportLink Object”. Gets an item from the Collection.	Read only	None
Parent	“SubreportObject Object”. Gets reference to the subreport link’s parent object.	Read only	None

SubreportLinks Collection Methods

The following methods are discussed in this section:

- “Add Method (SubreportLinks Collection)” on page 158
- “Delete Method (SubreportLinks Collection)” on page 159

Add Method (SubreportLinks Collection)

Examples

“How to import a subreport and link to a main report” on page 193

“How to re-import a subreport and link to a main report” on page 194

Use Add method to add a “SubreportLink Object” to the Collection.

Syntax

Function Add (MainReportField, SubreportField) As SubreportLink

Parameters

Parameter	Description
MainReportField	Specifies the field in the main report that you want the new Object in the Collection to link.
SubreportField	Specifies the field in the subreport that you want the new Object in the Collection to link.

Returns

Returns a “SubreportLink Object” member of the Collection.

Delete Method (SubreportLinks Collection)

Use Delete method to remove a “SubreportLink Object” from the Collection.

Syntax

Sub Delete (index As Long)

Parameter

Parameter	Description
index	Specifies the index number of the Object in the Collection that you want to remove.

SubreportObject Object

The SubreportObject Object represents a subreport placed in a report. A subreport is a free-standing or linked report found within the main report. This object provides properties for retrieving information on the subreport (for example, name, formatting options).

SubreportObject Object Properties

Examples

“How to format a subreport object” on page 190

Property	Description	Read/Write	Restriction in event handler
BackColor	OLE_COLOR. Gets or sets the object background color.	Read/Write	Can be written only when formatting idle or active.
BorderColor	OLE_COLOR. Gets or sets the object border color.	Read/Write	Can be written only when formatting idle or active.
BottomLineStyle	“CRLLineStyle”. Gets or sets the bottom line style.	Read/Write	Can be written only when formatting idle or active.
CanGrow	Boolean. Gets or sets the can grow option.	Read/Write	Can be written only when formatting idle or active.
CloseAtPage Break	Boolean. Gets or sets close border on page break.	Read/Write	Can be written only when formatting idle or active.
EnableOn Demand	Boolean. Gets the real-time subreport option.	Read only	None
HasDrop Shadow	Boolean. Gets or sets the border drop shadow option.	Read/Write	Can be written only when formatting idle or active.

Property	Description	Read/Write	Restriction in event handler
Height	Long. Gets or sets the object height, in twips.	Read/Write	Can be written only when formatting idle or active.
KeepTogether	Boolean. Gets or sets the keep object together option.	Read/Write	Can be written only when formatting idle or active.
Kind	"CROBJECTKind". Gets what kind of object (for example, box, cross-tab, field).	Read only	None
Left	Long. Gets or sets the object upper left position, in twips.	Read/Write	Can be written only when formatting idle or active.
LeftLineStyle	"CRLINEStyle". Gets or sets the left line style.	Read/Write	Can be written only when formatting idle or active.
Links	"REMARKS". Gets reference to the Collection.	Read Only	None
Name	String. Gets or sets the object name.	Read/Write	Can be written only when formatting idle or active.
Parent	"SECTION Object". Gets reference to the parent object.	Read only	None
RightLineStyle	"CRLINEStyle". Gets or sets the right line style.	Read/Write	Can be written only when formatting idle or active.
SubreportName	String. Gets or sets the name of the subreport.	Read/Write	Can be written only when formatting idle.
Suppress	Boolean. Gets or sets the object visibility option.	Read/Write	Can be written only when formatting idle or active.
Top	Long. Gets or sets the object upper top position, in twips.	Read/Write	Can be written only when formatting idle or active.
TopLineStyle	"CRLINEStyle". Gets or sets the top line style.	Read/Write	Can be written only when formatting idle or active.
Width	Long. Gets or sets the object width, in twips.	Read/Write	Can be written only when formatting idle or active.

SubreportObject Object Methods

The following methods are discussed in this section:

- "OpenSubreport Method (SubreportObject Object)" on page 161
- "ReImportSubreport Method (SubreportObject Object)" on page 161

OpenSubreport Method (SubreportObject Object)

Every subreport has a pointer to its parent report. Subreports hold on to their parent reports until they are destroyed. The OpenSubreport method opens a subreport contained in the report and returns a Report Object corresponding to the named subreport. It corresponds to PEOpenSubreport of the Crystal Report Engine. This method can be invoked only in FormattingIdle or FormattingActive mode.

Syntax

Function OpenSubreport () As Report

Returns

Returns a "SubreportObject Object".

ReImportSubreport Method (SubreportObject Object)

Examples

"How to re-import a subreport and link to a main report" on page 194

Subreports can be newly created in the main report or imported from an existing report file. If the subreport is imported from an existing report file, it can be re-imported at runtime using the ReImportSubreport method. When previewed, printed, or exported, a re-imported subreport will reflect all changes made to the formatting, grouping, and structure of the existing report file.

Syntax

Sub ReImportSubreport (pReimported as Boolean)

Parameter

Parameter	Description
pReimported	Boolean value indicating success or failure when re-importing the subreport. Will be set to TRUE if re-importing the subreport succeeds and FALSE if re-importing the subreport fails. See remarks below.

Remarks

pReimported is an output parameter. Create a variable to hold the result of pReimported and check the value of the variable to see if the method is successful.

SummaryFieldDefinition Object

The SummaryFieldDefinition Object represents the summary used in a cross-tab, group, or report.

SummaryFieldDefinition Object Properties

Property	Description	Read/Write	Restriction in event handler
FooterArea	"Area Object". Gets the area pair that the summary is in.	Read only	None
ForCrossTab	Boolean. Gets for-cross-tab.	Read only	None
HeaderArea	"Area Object". Gets the area pair that the summary is in.	Read only	None
HierarchicalSummaryType	"CRHierarchicalSummaryType". Gets or sets whether or not to calculate the summary across the hierarchy in a hierarchically grouped report.	Read/Write	Can be written only when formatting idle.
Kind	"CRFieldKind". Gets which kind of field (database, summary, formula, etc.).	Read only	None
Name	String. Gets the field definition unique formula name of the summary field for a group or report summary field. Cross-tab summaries do not have a unique name so an empty string will be returned.	Read only	None
NextValue	Variant. Gets the field next value.	Read only	Can be read only when top-level Report object is formatting.
NumberOfBytes	Integer. Gets the number of bytes required to store the field data in memory.	Read only	None
Parent	"Report Object". Gets reference to the parent object.	Read only	None
PreviousValue	Variant. Gets the field previous value.	Read only	Can be read only when top-level Report object is formatting.
SecondarySummarizedField	Object. Gets the secondary summarized field.	Read only	None
SummarizedField	Object. Gets the summarized field.	Read only	None

Property	Description	Read/Write	Restriction in event handler
SummaryOperationParameter	Long. Gets or sets the summary operation parameter.	Read/Write	Can be written only when formatting idle.
SummaryType	"CRSummaryType". Gets or sets the type of summary (for example, sum, average).	Read/Write	Can be written only when formatting idle.
Value	Variant. Gets the field current value.	Read only	Can be read only when top-level Report object is formatting active.
ValueType	"CRFieldValueType". Gets which type of value is found in the field.	Read only	None

SummaryFieldDefinition Object Methods

The following methods are discussed in this section:

- "SetSecondarySummarizedField Method (SummaryFieldDefinition Object)" on page 163
- "SetSummarizedField Method (SummaryFieldDefinition Object)" on page 163

SetSecondarySummarizedField Method (SummaryFieldDefinition Object)

Use SetSecondarySummarizedField method to set the secondary summarized field for a report.

Syntax

```
Sub SetSecondarySummarizedField (secondarySummariedField)
```

Parameter

Parameter	Description
secondarySummariedField	Specifies the field that you want to set.

SetSummarizedField Method (SummaryFieldDefinition Object)

Syntax

```
Sub SetSummarizedField (SummarizedField)
```

Parameter

Parameter	Description
SummarizedField	Specifies the field that you want to set.

SummaryFieldDefinitions Collection

The SummaryFieldDefinitions Collection is a collection of summary field definitions. Can be from either CrossTabObject Object or Report Object. Access a specific SummaryFieldDefinition Object in the collection using the Item property.

SummaryFieldDefinitions Collection Properties

Property	Description	Read/Write	Restriction in event handler
Count	Long. Gets the number of summary fields in the collection.	Read only	None
Item (index As Long)	“SummaryFieldDefinition Object” . Gets an item from the Collection. Item has an index parameter that is a numeric, 1-based index (for example, Item (1)). The items in the collection are indexed in the order they were added to the report.	Read only	None
Parent	“Report Object” . Gets reference to the parent object.	Read only	None

Remarks

Instead of using the Item property as shown, you can reference a summary field directly, for example, SummaryFieldDefinitions(1).

SummaryFieldDefinitions Collection Methods

The following methods are discussed in this section:

- [“Add Method \(SummaryFieldDefinitions Collection\)”](#) on page 164
- [“Delete Method \(SummaryFieldDefinitions Collection\)”](#) on page 165

Add Method (SummaryFieldDefinitions Collection)

Use Add method to add a [“SummaryFieldDefinition Object”](#) to the Collection.

Syntax

```
Function Add (groupLevel As Long, Field, _
    SummaryType As CRSummaryType, [secondSummaryFieldOrFactor] _
    ) As SummaryFieldDefinition
```

Parameters

Parameter	Description
groupLevel	Specifies the group level for the summary.
Field	Specifies the summary field.

Parameter	Description
SummaryType	"CRSummaryType". Specifies the summary type.
[secondSummaryField OrFactor]	Specifies the second summary field or factor.

Returns

Returns a "SummaryFieldDefinition Object" member of the Collection.

Delete Method (SummaryFieldDefinitions Collection)

Use delete method to delete a "SummaryFieldDefinition Object" from the Collection.

Syntax

Sub Delete (index)

Parameter

Parameter	Description
index	Specifies the index number of the object that you want to delete from the Collection.

TableLink Object

The TableLink Object provides information about database table links.

TableLink Object Properties

Examples

"How to add and delete table links" on page 196

Property	Description	Read/Write	Restriction in event handler
DestinationFields	"DatabaseFieldDefinition Object". Gets reference to table link destination field definitions Collection.	Read only	None
DestinationTable	"DatabaseTable Object". Gets reference to the table link destination table.	Read only	None
JoinType	"CRLinkJoinType". Gets the table link join type.	Read only	None
LookupType	"CRLinkLookUpType". Gets table link lookup type.	Read only	None

Property	Description	Read/Write	Restriction in event handler
Parent	"Database Object". Gets reference to the parent object.	Read only	None
SourceFields	"DatabaseFieldDefinition Object". Gets the table link source field definitions.	Read only	None
SourceTable	"DatabaseTable Object". Gets table link source table.	Read only	None

TableLinks Collection

The TableLinks Collection contains the "TableLink Object" Objects associated with the report.

TableLinks Collection Properties

Property	Definition	Read/Write	Restriction in event handler
Count	Long. Gets a count of the collection items.	Read only	None
Item (index As Long)	"TableLink Object". Gets an item from the Collection.	Read only	None
Parent	"Database Object". Gets reference to the parent object.	Read only	None

TableLinks Collection Methods

The following methods are discussed in this section:

- "Add Method (TableLinks Collection)" on page 166
- "Delete Method (TableLinks Collection)" on page 167

Add Method (TableLinks Collection)

Examples

"How to add and delete table links" on page 196

Use Add method to add a "TableLink Object" to the Collection.

Syntax

```
Function Add (psrcTable As DatabaseTable, pDestTable As DatabaseTable, _
    srcFields, destFields, _
    JoinType As CRLinkJoinType, LookupType As CRLinkLookUpType, _
    PartialMatchEnabled As Boolean, indexInUse As Integer) As TableLink
```

Parameters

Parameter	Description
psrcTable	“DatabaseTable Object”. Specifies the source database table.
pDestTable	“DatabaseTable Object”. Specifies the destination database table.
srcFields	Specifies the source fields.
destFields	Specifies the destination fields.
JoinType	“CRLinkJoinType”. Specifies the link join type.
LookupType	“CRLinkLookUpType”. Specifies the link lookup type.
PartialMatchEnabled	Specifies whether the PartialMatchEnabled option has been set.
indexInUse	Specifies index in use.

Returns

Returns a “TableLink Object” member of the Collection.

Delete Method (TableLinks Collection)

Examples

“How to add and delete table links” on page 196

Use Delete method to remove a “TableLink Object” from the Collection.

Syntax

```
Sub Delete (index As Long)
```

Parameter

Parameter	Description
index	Specifies the index number of the item that you want to remove from the Collection.

TextObject Object

The Text Object represents a text object found in a report. This object provides properties for retrieving information and setting options for a text object in your report.

TextObject Object Properties

Examples

“How to format a text object” on page 192

Property	Description	Read/Write	Restriction in event handler
BackColor	OLE_COLOR. Gets or sets the object background color.	Read/Write	Can be written only when formatting idle or active.
BorderColor	OLE_COLOR. Gets or sets the object border color.	Read/Write	Can be written only when formatting idle or active.
BottomLineStyle	“CRLLineStyle”. Gets or sets the bottom line style.	Read/Write	Can be written only when formatting idle or active.
CanGrow	Boolean. Gets or sets the can grow option.	Read/Write	Can be written only when formatting idle or active.
CharacterSpacing	Long. Gets or sets the character spacing.	Read/Write	Can be written only when formatting idle.
CloseAtPage Break	Boolean. Gets or sets th close border on page break option.	Read/Write	Can be written only when formatting idle or active.
Condition Formula	String. Gets or sets the condition formula. The condition formula can be based on recurring records or on summary fields, but it cannot be based on print-time fields, such as running totals or print-time formulas. Condition formulas cannot have shared variables.	Read/Write	Can be written only when formatting idle.
CssClass	String. Gets or sets the cascading style sheet Class.	Read/Write	Can be written only when formatting idle.
FieldElements	“SetUnboundFieldSource Method (FieldObject Object)”. Gets the field elements Collection.	Read Only	None
FirstLineIndent	Long. Gets or sets first line indent, in twips.	Read/Write	Can be written only when formatting idle.
Font	IFontDisp. Gets or sets the standard OLE font of the text object.	Read/Write	Can be written only when formatting idle or active.
HasDropShadow	Boolean. Gets or sets the border drop shadow option.	Read/Write	Can be written only when formatting idle or active.
Height	Long. Gets or sets the object height, in twips.	Read/Write	Can be written only when formatting idle or active.
HorAlignment	“CRAAlignment”. Gets or sets the horizontal alignment.	Read/Write	Can be written only when formatting idle or active.
KeepTogether	Boolean. Gets or sets the keep area together option.	Read/Write	Can be written only when formatting idle or active.

Property	Description	Read/Write	Restriction in event handler
Kind	"CROBJECTKind". Gets what kind of object (for example, box, cross-tab, field).	Read only	None
Left	Long. Gets or sets the object upper left position, in twips.	Read/Write	Can be written only when formatting idle or active.
LeftIndent	Long. Gets or sets left indent, in twips.	Read/Write	Can be written only when formatting idle.
LeftLineStyle	"CRLINEStyle". Gets or sets the left line style.	Read/Write	Can be written only when formatting idle or active.
LineSpacing	Double. Gets the line spacing.	Read only	Can be written only when formatin idle.
LineSpacingType	"CRLINESpacingType". Gets the line spacing type.	Read only	
MaxNumberOfLines	Integer. Gets or sets the maximum number of line for a string memo field.	Read/Write	Can be written only when formatting idle or active.
Name	String. Gets or sets area name.	Read/Write	Can be written only when formatting idle.
Parent	"Section Object". Gets reference to the parent object.	Read only	None
RightIndent	Long. Gets or sets the right indent.	Read/Write	Can be written only when formatting idle.
RightLineStyle	"CRLINEStyle". Gets or sets the right line style.	Read/Write	Can be written only when formatting idle or active.
Suppress	Boolean. Gets or sets object visibility.	Read/Write	Can be written only when formatting idle or active.
SuppressIfDuplicated	Boolean. Gets or sets the suppress if duplicated option.	Read/Write	Can be written only when formatting idle or active.
Text	String. Gets the text within the text object. If it has embedded fields, [] is used to represent the embedded field.	Read only	None
TextColor	OLE_COLOR. Gets or sets the object text color.	Read/Write	Can be written only when formatting idle or active.
TextRotationAngle	"CRRotationAngle". Gets or sets text rotation angle.	Read/Write	Can be written only when formatting idle or active.
Top	Long. Gets or sets the object upper top position, in twips.	Read/Write	Can be written only when formatting idle or active.
TopLineStyle	"CRLINEStyle". Gets or sets the top line style.	Read/Write	Can be written only when formatting idle or active.
Width	Long. Gets or sets the object width, in twips.	Read/Write	Can be written only when formatting idle or active.

TextObject Object Methods

The following methods are discussed in this section:

- “SetLineSpacing Method (TextObject Object)” on page 170
- “SetText Method (TextObject Object)” on page 170

SetLineSpacing Method (TextObject Object)

Use the SetLineSpacing method to set the line spacing for a “TextObject Object”. The information passed will be used during formatting.

Syntax

```
Sub SetLineSpacing (LineSpacing As Double, _
    LineSpacingType As CRLLineSpacingType)
```

Parameters

Parameter	Description
LineSpacing	Specifies the line spacing.
LineSpacingType	“CRLLineSpacingType”. Specifies the line spacing type.

SetText Method (TextObject Object)

Use the SetText method to set the text object to return the specified text. This method can be invoked only in the FormattingIdle or FormattingActive mode.

Syntax

```
Sub SetText (pText As String)
```

Parameter

Parameter	Description
pText	Specifies the text string for the Object. See Remarks below.

Remarks

When the SetText method is called within the sections Format event handler, the text parameter passed should not be dependent upon some state that the event handler may be tracking. For example, the event handler may hold a count and increment it each time the Format event is fired. This could should not be used directly or indirectly to create the SetText text parameter.

Examples

The following examples assume the developer is already familiar with using the RDC in Visual Basic and how to create reports. Unless noted otherwise most reports are created in the Visual Basic IDE and are named CrystalReport1. Each example gives a brief description on how the report or reports are created including tables used, grouping, charts and so on. If you need help in creating a report see the *Crystal Reports Online Help (crw.chm)*. If you need help with using the RDC see "Quick Start for using the RDC" and "RDC Programming" in the *Crystal Reports Developer's Help (CrystalDevHelp.chm)*.

- "How to connect to a Microsoft Access database" on page 171
- "How to connect to a secured Microsoft Access database" on page 173
- "How to connect to an ODBC data source" on page 175
- "How to connect to an OLEDB data source" on page 177
- "How to change the data source" on page 178
- "How to list the connection properties and connect to a report" on page 179
- "How to format a blob field object" on page 182
- "How to format a box object" on page 184
- "How to format a graph object (chart)" on page 185
- "How to format a map object" on page 187
- "How to format an olap grid object" on page 188
- "How to format an OLE object" on page 189
- "How to format a subreport object" on page 190
- "How to format a text object" on page 192
- "How to import a subreport and link to a main report" on page 193
- "How to re-import a subreport and link to a main report" on page 194
- "How to add and delete table links" on page 196
- "How to add, delete and format groups in a crosstab" on page 197
- "How to format a group in the section format event" on page 200

How to connect to a Microsoft Access database

This example demonstrates three methods for changing the location of an Access database. In each example CrystalReport1 is connected to the xtreme.mdb database found in the \Program Files\Crystal Decisions\Crystal Reports 9\Samples\En\Databases folder. The report is using the Customer table. A copy of xtreme.mdb is placed in a new directory, and a copy of the Customer table (NewCustomer) is added to xtreme.mdb.

Example 1

This example uses the Location property of the [DatabaseTable Object](#), to change the location of the database.

```

' Create a new instance of the report.
Dim Report As New CrystalReport1

Private Sub Form_Load()
' Declare a DatabaseTable object.
Dim DBTable As CRAXDRT.DatabaseTable

' Get the first table in the report.
Set DBTable = Report.Database.Tables(1)

' Set the new location of the database.
DBTable.Location = "C:\databases\xtreme.mdb"

Screen.MousePointer = vbHourglass
' Set the report source of the viewer and view the report.
CRViewer91.ReportSource = Report
CRViewer91.ViewReport
Screen.MousePointer = vbDefault

End Sub

```

Example 2

This example uses the Location property of the **ConnectionProperty Object** to change the location of the database.

```

' Create a new instance of the report.
Dim Report As New CrystalReport1

Private Sub Form_Load()
' Declare a ConnectionProperty object.
Dim CPPProperty As CRAXDRT.ConnectionProperty
' Declare a DatabaseTable object.
Dim DBTable As CRAXDRT.DatabaseTable

' Get the first table in the report.
Set DBTable = Report.Database.Tables(1)

' Get the "Database Name" property from the
' ConnectionProperties collection.
Set CPPProperty = DBTable.ConnectionProperties("Database Name")

' Set the new location of the database.
CPPProperty.Value = "C:\databases\xtreme.mdb"

Screen.MousePointer = vbHourglass
' Set the report source of the viewer and view the report.
CRViewer91.ReportSource = Report
CRViewer91.ViewReport
Screen.MousePointer = vbDefault

End Sub

```

Example 3

This example uses the **SetTableLocation Method (DatabaseTable Object)** to change the location of the database and the name of the table.

```
' Create a new instance of the report.
Dim Report As New CrystalReport1

Private Sub Form_Load()
' Declare a DatabaseTable object.
Dim DBTable As CRAXDRT.DatabaseTable

' Get the first table in the report.
Set DBTable = Report.Database.Tables(1)

' Set the new database location and the new table name.
DBTable.SetTableLocation "C:\databases\xtreme.mdb", NewCustomer, ""

Screen.MousePointer = vbHourglass
' Set the report source of the viewer and view the report.
CRViewer91.ReportSource = Report
CRViewer91.ViewReport
Screen.MousePointer = vbDefault

End Sub
```

How to connect to a secured Microsoft Access database

This example demonstrates three methods for changing the location of an Access database and setting the password to the database. In each example **CrystalReport1** is connected to the **xtreme.mdb** database found in the **\Program Files\Crystal Decisions\Crystal Reports 9\Samples\En\Databases** folder. The report is using the **Customer** table. A copy of **xtreme.mdb** is placed in a new directory, and a copy of the **Customer** table (**NewCustomer**) is added to **xtreme.mdb**. A password is added to the database to make it a secured Access database.

Example 1

This example uses the **Location** property of the **DatabaseTable Object** to change the location of the database, and the **ConnectionProperty Object** to set the password.

```
' Create a new instance of the report.
Dim Report As New CrystalReport1

Private Sub Form_Load()
' Declare a ConnectionProperty object.
Dim CPPProperty As CRAXDRT.ConnectionProperty
' Declare a DatabaseTable object.
Dim DBTable As CRAXDRT.DatabaseTable
```

```

' Get the first table in the report.
Set DBTable = Report.Database.Tables(1)

' Set the new location of the database.
DBTable.Location = "C:\databases\xtreme.mdb"

' Get the "Database Password" property from the
' ConnectionProperties collection.
Set CPProperty = DBTable.ConnectionProperties("Database Password")

' Set the database password.
CPProperty.Value = "password"

Screen.MousePointer = vbHourglass
' Set the report source of the viewer and view the report.
CRViewer91.ReportSource = Report
CRViewer91.ViewReport
Screen.MousePointer = vbDefault

End Sub

```

Example 2

This example uses the **ConnectionProperty Object** to change the location of the database and to set the password.

```

' Create a new instance of the report.
Dim Report As New CrystalReport1

Private Sub Form_Load()
' Declare a ConnectionProperty object.
Dim CPProperty As CRAXDRT.ConnectionProperty
' Declare a DatabaseTable object.
Dim DBTable As CRAXDRT.DatabaseTable

' Get the first table in the report.
Set DBTable = Report.Database.Tables(1)

' Get the "Database Name" property from the
' ConnectionProperties collection.
Set CPProperty = DBTable.ConnectionProperties("Database Name")

' Set the new location of the database.
CPProperty.Value = "C:\databases\xtreme.mdb"

' Get the "Database Password" property from the
' ConnectionProperties collection.
Set CPProperty = DBTable.ConnectionProperties("Database Password")

' Set the database password.
CPProperty.Value = "password"

```



```

Screen.MousePointer = vbHourglass
' Set the report source of the viewer and view the report.
CRViewer91.ReportSource = Report
CRViewer91.ViewReport
Screen.MousePointer = vbDefault

End Sub

```

Example 3

This example uses the **SetTableLocation Method (DatabaseTable Object)** to change the location of the database and the name of the table, and the **ConnectionProperty Object** to set the password.

```

' Create a new instance of the report.
Dim Report As New CrystalReport1

Private Sub Form_Load()
' Declare a ConnectionProperty object.
Dim CPPProperty As CRAXDRT.ConnectionProperty
' Declare a DatabaseTable object.
Dim DBTable As CRAXDRT.DatabaseTable

' Get the first table in the report.
Set DBTable = Report.Database.Tables(1)

' Set the new database location and the new table name.
DBTable.SetTableLocation "C:\databases\xtreme.mdb", NewCustomer, ""

' Get the "Database Password" property from the
' ConnectionProperties collection.
Set CPPProperty = DBTable.ConnectionProperties("Database Password")

' Set the database password.
CPPProperty.Value = "password"

Screen.MousePointer = vbHourglass
' Set the report source of the viewer and view the report.
CRViewer91.ReportSource = Report
CRViewer91.ViewReport
Screen.MousePointer = vbDefault

End Sub

```

How to connect to an ODBC data source

This example demonstrates how to connect to an ODBC data source by using the **ConnectionProperty Object**, and how to change the table name by using the **Location** property of the **DatabaseTable Object**. CrystalReport1 is created using an ODBC data source connected to the pubs database on Microsoft SQL Server. The report is based off the authors table.

```

' Create a new instance of the report.
Dim Report As New CrystalReport1

Private Sub Form_Load()
' Declare a ConnectionProperty object.
Dim CPPProperty As CRAXDRT.ConnectionProperty
' Declare a DatabaseTable object.
Dim DBTable As CRAXDRT.DatabaseTable

' Get the first table in the report.
Set DBTable = Report.Database.Tables(1)

' Get the "DSN" property from the
' ConnectionProperties collection.
Set CPPProperty = DBTable.ConnectionProperties("DSN")

' Set the ODBC data source name.
' Note: You do not need to set this property if
' you are using the same ODBC data source.
CPPProperty.Value = "ODBCSQL"

' Get the "User ID" property from the
' ConnectionProperties collection.
Set CPPProperty = DBTable.ConnectionProperties("User ID")

' Set the user name.
' Note: You do not need to set this property if
' you are using the same user name.
CPPProperty.Value = "username"

' Get the "Password" property from the
' ConnectionProperties collection.
' Note: You must always set the password if one is
' required.
Set CPPProperty = DBTable.ConnectionProperties("Password")

' Set the password.
CPPProperty.Value = "password"

' Get the "Database" property from the
' ConnectionProperties collection.
Set CPPProperty = DBTable.ConnectionProperties("Database")

' Set the database.
CPPProperty.Value = "master"

' Set the table name.
DBTable.Location = "authors2"

Screen.MousePointer = vbHourglass
' Set the report source of the viewer and view the report.
CRViewer91.ReportSource = Report
CRViewer91.ViewReport

```

```
Screen.MousePointer = vbDefault
End Sub
```

How to connect to an OLEDB data source

This example demonstrates how to connect to an OLEDB (ADO) data source, change the data source, and change the database by using the **ConnectionProperty Object**, as well as how to change the table name by using the **Location** property of the **DatabaseTable Object**. CrystalReport1 is created using an ODBC data source connected to the pubs database on Microsoft SQL Server. The report is based off the authors table.

```
' Create a new instance of the report.
Dim Report As New CrystalReport1

Private Sub Form_Load()
' Declare a ConnectionProperty object.
Dim CPPProperty As CRAXDRT.ConnectionProperty
' Declare a DatabaseTable object.
Dim DBTable As CRAXDRT.DatabaseTable

' Get the first table in the report.
Set DBTable = Report.Database.Tables(1)

' Get the "Data Source" property from the
' ConnectionProperties collection.
Set CPPProperty = DBTable.ConnectionProperties("Data Source")

' Set the new data source (server name).
' Note: You do not need to set this property if
' you are using the same data source.
CPPProperty.Value = "Server2"

' Get the "User ID" property from the
' ConnectionProperties collection.
Set CPPProperty = DBTable.ConnectionProperties("User ID")

' Set the user name.
' Note: You do not need to set this property if
' you are using the same user name.
CPPProperty.Value = "UserName"

' Get the "Password" property from the
' ConnectionProperties collection.
Set CPPProperty = DBTable.ConnectionProperties("Password")

' Set the password.
' Note: You must always set the password if one is
' required.
CPPProperty.Value = "Password"
```

```

' Get the "Initial Catalog" (database name) property from the
' ConnectionProperties collection.
' Note: You do not need to set this property if
' you are using the same database.
Set CPProperty = DBTable.ConnectionProperties("Initial Catalog")

' Set the new database name.
CPProperty.Value = "master"

' Set the new table name.
DBTable.Location = "authors2"

Screen.MousePointer = vbHourglass
' Set the report source of the viewer and view the report.
CRViewer91.ReportSource = Report
CRViewer91.ViewReport
Screen.MousePointer = vbDefault
End Sub

```

How to change the data source

This example demonstrates how to change the data source from native Access to an OLEDB (ADO) data source by using the **ConnectionProperty Object**, as well as how to change the table name by using the Location property of the **DatabaseTable Object**. CrystalReport1 is connected to the xtreme.mdb database found in the \Program Files\Crystal Decisions\Crystal Reports 9\Samples\En\Databases folder. The report is using the Customer table. A copy of the Customer table is added to the pubs database on Microsoft SQL Server.

```

' Create a new instance of the report.
Dim Report As New CrystalReport1

Private Sub Form_Load()
' Declare a ConnectionProperties collection.
Dim CPProperties As CRAXDRT.ConnectionProperties
' Declare a DatabaseTable object.
Dim DBTable As CRAXDRT.DatabaseTable

' Get the first table in the report.
Set DBTable = Report.Database.Tables(1)

' Get the collection of connection properties.
Set CPProperties = DBTable.ConnectionProperties

' Change the database DLL used by the report from
' native Access (crdb_dao.dll) to ADO/OLEDB (crdb_ado.dll).
DBTable.DLLName = "crdb_ado.dll"

' The connection property bags contain the name and value
' pairs for the native Access DLL (crdb_dao.dll). So we need

```

```

' to clear them, and then add the name and value pairs that
' are required to connect to the OLEDB data source.

' Clear all the ConnecioProperty objects from the collection.
CPPProperties.DeleteAll

' Add the name value pair for the provider.
CPPProperties.Add "Provider", "SQLOLEDB"

' Add the name value pair for the data source (server).
CPPProperties.Add "Data Source", "ServerA"

' Add the name value pair for the database.
CPPProperties.Add "Initial Catalog", "pubs"

' Add the name value pair for the user name.
CPPProperties.Add "User ID", "UserName"

' Add the name value pair for the password.
CPPProperties.Add "Password", "password"

' Set the table name.
DBTable.Location = "Customer"

Screen.MousePointer = vbHourglass
' Set the report source of the viewer and view the report.
CRViewer91.ReportSource = Report
CRViewer91.ViewReport
Screen.MousePointer = vbDefault
End Sub

```

How to list the connection properties and connect to a report

This example demonstrates how to list the connection property information by using the **ConnectionProperty Object**. The user can then set the values for the connection properties and display the report. The example consists of two forms and a report designed in the Visual Basic IDE using the RDC. The first form allows the user to view and set the connection properties. The second form displays the report.

To use the example

- 1 Create a standard application and add an additional form.
- 2 Create or import a report using the RDC.
- 3 Add the following controls to Form1:
 - Combo List Box (Combo1)
In the property window set Style to "2 - Dropdown List".
Combo1 holds the list of connection property names.

- Combo List Box (Combo2)
In the property window set Style to “1 - Simple Combo”.
Combo2 holds the list of connection property values.
 - Button (Command1)
Command1 sets the connection property values.
 - Button (Command2)
Command2 hides this form and loads the second form.
- 4 Cut and paste the code that is listed under Form1 below into the code window for Form1 in your application.
 - 5 Add the following controls to Form2:
 - Crystal Reports Viewer Control (CRViewer91)
 - 6 Cut and paste the code that is listed under Form2 below into the code window for Form2 in your application.
 - 7 Run the application.
 - 8 With the application running the user can:
 - Select the connection property from Combo1.
 - Change the value in the Combo2.
 - Click Command1 to save the changes to the value.
 - Click Command2 to display the report in Form 2.

The first form (Form1)

This is the start up form and displays the name and value pairs for each connection property.

```
' Create a new instance of the report.
Public Report As New CrystalReport1
' Declare a ConnectionProperty object.
Dim CPProperty As CRAXDRT.ConnectionProperty
' Declare a DatabaseTable object.
Dim DBTable As CRAXDRT.DatabaseTable

' When an item in the ComboBox1(connection property name)
' is selected the corresponding item in ComboBox2
' (connection property value) is displayed.
Private Sub Combo1_Click()
    Combo2.ListIndex = Combo1.ListIndex
End Sub

'Clicking this button sets the connection property
' value.
Private Sub Command1_Click()
    ' This variable holds the new value for the selected
    ' connection property.
    Dim NewValue As String
```

```

' Get the selected connection property.
Set CPPProperty = DBTable.ConnectionProperties(Combo1.Text)
' Get and set the new value for the connection property.
NewValue = Combo2.Text
CPPProperty.Value = NewValue
' Update the combo box with the new value.
Combo2.RemoveItem Combo1.ListIndex
Combo2.AddItem NewValue, Combo1.ListIndex
End Sub

' Clicking this button displays the report
' and hides this form.
Private Sub Command2_Click()
Form2.Show
Form1.Hide
End Sub

Private Sub Form_Load()
' Get the first table in the report.
Set DBTable = Report.Database.Tables(1)

' Create a list of value and name pairs for each
' connection property in the collection.
For Each CPPProperty In DBTable.ConnectionProperties
' Add the name of the connection property to the
' first combo box.
Combo1.AddItem CPPProperty.Name
' Add the value of the connection property to the
' second combo box. Password is a write only
' property so set it to an empty string.
If InStr(1, CPPProperty.Name, "Password", 1) > 0 Then
Combo2.AddItem ""
Else
Combo2.AddItem CPPProperty.Value
End If
Next CPPProperty

' Set both combo boxes to the first item.
Combo1.ListIndex = 0
Combo2.ListIndex = 0
End Sub

Private Sub Form_Unload(Cancel As Integer)
' Destroy the Report object.
Set Report = Nothing
End Sub

```

The second form (Form2)

This form displays the report set in Form1.

```
Private Sub Form_Load()
    Screen.MousePointer = vbHourglass
    ' Set the report source of the viewer and view the report.
    CRViewer91.ReportSource = Form1.Report
    CRViewer91.ViewReport
    Screen.MousePointer = vbDefault
End Sub

Private Sub Form_Resize()
    ' Resize the viewer to fit the form.
    CRViewer91.Top = 0
    CRViewer91.Left = 0
    CRViewer91.Height = ScaleHeight
    CRViewer91.Width = ScaleWidth
End Sub

Private Sub Form_Unload(Cancel As Integer)
    ' Show Form1.
    Form1.Show
End Sub
```

How to format a blob field object

This example gets a blob field object and then formats the blob field object through the **BlobFieldObject** properties. CrystalReport1 is connected to the **xtrme.mdb** database found in the **\Program Files\Crystal Decisions\Crystal Reports 9\Samples\En\Databases** folder. The report is using the **Employees** table and the **Photo** field is added to the details section.

```
' Create a new instance of the report.
Dim oReport As New CrystalReport1

Private Sub Form_Load()
    ' Declare an BlobFieldObject object.
    Dim oBlobfield As CRAXDRT.BlobFieldObject
    ' Declare a generic Object.
    Dim oObject As Object
    ' Declare a Section object.
    Dim oSection As CRAXDRT.Section
    ' This variable is set to true once the
    ' blob field object is found.
    Dim bBlob As Boolean
    bBlob = False
```



```

' Search for all the report objects in each section.
For Each oSection In oReport.Sections
    For Each oObject In oSection.ReportObjects
        ' Find the first blob field object.
        If oObject.Kind = crBlobFieldObject Then
            ' Get the blob field object and exit the loop.
            Set oBlobfield = oObject
            bBlob = True
            Exit For
        End If
    Next oObject
    If bBlob Then Exit For
Next oSection

' Format the blobfield object.
With oBlobfield
    .BackColor = vbBlue
    .BorderColor = vbGreen
    .BottomCropping = 250
    .TopCropping = 250
    .LeftCropping = 250
    .RightCropping = 250
    .BottomLineStyle = crLSDoubleLine
    .TopLineStyle = crLSDashLine
    .LeftLineStyle = crLSSingleLine
    .RightLineStyle = crLSDotLine
    .HasDropShadow = True
    .CloseAtPageBreak = True
    .KeepTogether = False
    .Height = 2000
    .Width = 2000
    .Top = 5000
    .Left = 1000
    .Suppress = False
    ' X and Y scaling gets or sets width scaling factor.
    ' For example 1 means 100%, 2 means 200%, 0.5 means 50%,
    ' and so on. The allowable range is from 0.01 to 100.
    ' This will change the actual height and width of the object.
    .XScaling = 2
    .YScaling = 2
End With

' Set the report source of the viewer and view the report.
CRViewer91.ReportSource = oReport
CRViewer91.ViewReport
End Sub

```

How to format a box object

This example gets a box object and then formats the box object through the **BoxObject** properties. CrystalReport1 is connected to the xtreme.mdb database found in the \Program Files\Crystal Decisions\Crystal Reports 9\Samples\En\Databases folder. The report is using the Customer table and a box is inserted into the report.

```
' Create a new instance of the report.
Dim oReport As New CrystalReport1

Private Sub Form_Load()
    ' Declare an BoxObject object.
    Dim oBox As CRAXDRT.BoxObject
    ' Declare a generic Object.
    Dim oObject As Object
    ' Declare a Section object.
    Dim oSection As CRAXDRT.Section
    ' This variable is set to true once the box object is found.
    Dim bBox As Boolean
    bBox = False

    ' Search for all the report objects in each section.
    For Each oSection In oReport.Sections
        For Each oObject In oSection.ReportObjects
            ' Find the first box object.
            If oObject.Kind = crBoxObject Then
                ' Get the box object and exit the loop.
                Set oBox = oObject
                bBox = True
                Exit For
            End If
        Next oObject
        If bBox Then Exit For
    Next oSection

    ' Format the box object.
    With oBox
        .Bottom = 250
        ' BottomRightSection refers to the report section
        ' containing the bottom right corner
        ' of this box object.
        .BottomRightSection.BackColor = vbCyan
        .CloseAtPageBreak = True
        .CornerEllipseHeight = 500
        .CornerEllipseWidth = 500
        .ExtendToBottomOfSection = True
        .FillColor = vbMagenta
        ' DropShadow cannot be set to True for a
        ' rounded corner box object.
        .HasDropShadow = False
    End With
End Sub
```

```

.Left = 250
.LineColor = vbBlue
.LineStyle = crLSDotLine
.Right = 7400
.Suppress = False
.Top = 100
End With

' Set the report source of the viewer and view the report.
CRViewer91.ReportSource = oReport
CRViewer91.ViewReport
End Sub

```

How to format a graph object (chart)

This example gets a graph object and then formats the graph object through the **GraphObject Object** properties. CrystalReport1 is connected to the xtreme.mdb database found in the \Program Files\Crystal Decisions\Crystal Reports 9\Samples\En\Databases folder. The report is using the Customer table and a chart is inserted into the report. See “Creating Charts” in the *Crystal Reports Online Help* (*crw.chm*)

```

' Create a new instance of the report.
Dim oReport As New CrystalReport1

Private Sub Form_Load()
' Declare an GraphObject object.
Dim oGraph As CRAXDRT.GraphObject
' Declare a generic Object.
Dim oObject As Object
' Declare a Section object.
Dim oSection As CRAXDRT.Section
' This variable is set to true once the graph object is found.
Dim bGraph As Boolean
bGraph = False

' Search for all the report objects in each section.
For Each oSection In oReport.Sections
    For Each oObject In oSection.ReportObjects
        ' Find the first graph object.
        If oObject.Kind = crGraphObject Then
            ' Get the graph object and exit the loop.
            Set oGraph = oObject
            bGraph = True
            Exit For
        End If
    Next oObject
    If bGraph Then Exit For
Next oSection

```

```

' Format the graph object.
With oGraph
    .AutoRangeData2Axis = True
    .AutoRangeDataAxis = True
    .BackColor = vbYellow
    .BarSize = crAverageBarSize
    .BorderColor = vbRed
    .BottomLineStyle = crLSDashLine
    .CloseAtPageBreak = True
    .Data2AxisDivisionMethod = crAutomaticDivision
    .Data2AxisGridline = crMajorAndMinorGridlines
    .Data2AxisNumberFormat = crCurrencyTwoDecimal
    .Data2LabelFont.Bold = True
    .Data2Title = "Not In Use"
    .Data2TitleFont.Bold = False
    .DataAxisDivisionMethod = crAutomaticDivision
    .DataAxisGridline = crMajorAndMinorGridlines
    .DataAxisNumberFormat = crCurrencyThousands
    .DataLabelFont.Underline = True
    .DataPoint = crShowValue
    .DataTitle = "DataTitle"
    .DataTitleFont.Italic = True
    .DataValueNumberFormat = crCurrencyNoDecimal
    .EnableAutoScaleData2Axis = True
    .EnableAutoScaleDataAxis = True
    .EnableShowLegend = True
    .FootNote = "FootNote"
    .FootnoteFont.Size = 14
    .GraphColor = crColorGraph
    .GraphDirection = crHorizontalGraph
    .GraphType = crFaked3DStackedBarGraph
    .GroupAxisGridline = crMajorAndMinorGridlines
    .GroupLabelFont.Size = 8
    .GroupsTitle = "GroupsTitle"
    .GroupTitleFont.Weight = 999
    .HasDropShadow = True
    .IsData2TitleByDefault = True
    .IsDataTitleByDefault = True
    .IsFootnoteByDefault = False
    .IsGroupsTitleByDefault = False
    .IsSeriesTitleByDefault = False
    .IsSubTitleByDefault = False
    .IsTitleByDefault = True
    .KeepTogether = True
    .Left = 200
    .LeftLineStyle = crLSDoubleLine
    .LegendFont.Italic = True
    .LegendLayout = crBothLayout
    .LegendPosition = crPlaceLeft
    .MarkerShape = crTriangleShape
    .MaxData2AxisValue = 999999
    .MaxDataAxisValue = 999999
    .MinData2AxisValue = -999999

```

```

.MinDataAxisValue = -999999
.Name = "ThisIsGraph"
.PieSize = crLargePieSize
.RightLineStyle = crLSDotLine
.SeriesAxisGridline = crMajorAndMinorGridlines
.SeriesLabelFont.Strikethrough = True
.SeriesTitle = "SeriesTitle"
.SeriesTitleFont.Underline = True
.SliceDetachment = crSmallestSlice
.SubTitle = "SubTitle"
.SubTitleFont.Italic = True
.Suppress = False
.Title = "Title"
.TitleFont.Size = 18
.Top = 1100
.TopLineStyle = crLSDotLine
.ViewingAngle = crFewSeriesView
.Width = 9000
End With

' Set the report source of the viewer and view the report.
CRViewer91.ReportSource = oReport
CRViewer91.ViewReport
End Sub

```

How to format a map object

This example gets a map object then formats the map object through the **MapObject** properties. Create a report in the Crystal Report Designer using the **xtreme.mdb** database found in the **\Program Files\Crystal Decisions\Crystal Reports 9\Samples\En\Databases** folder. Add a map to the report and then import the report into a Visual Basic application. For maps see "Creating Maps" in the *Crystal Reports Online Help (crw.chm)* and for importing reports see "Open an existing report" on page 57.

```

' Create a new instance of the report.
Dim oReport As New CrystalReport1

Private Sub Form_Load()
' Declare an MapObject object.
Dim oMap As CRAXDRT.MapObject
' Declare a generic Object.
Dim oObject As Object
' Declare a Section object.
Dim oSection As CRAXDRT.Section
' This variable is set to true once the map object is found.
Dim bMap As Boolean
bMap = False

' Search for all the report objects in each section.
For Each oSection In oReport.Sections

```

```

For Each oObject In oSection.ReportObjects
    ' Find the first map object.
    If oObject.Kind = crMapObject Then
        ' Get the map object and exit the loop.
        Set oMap = oObject
        bMap = True
        Exit For
    End If
Next oObject
If bMap Then Exit For
Next oSection

' Format the map object.
With oMapObject
    .BackColor = vbGreen
    .BorderColor = vbRed
    .BottomLineStyle = crLSDoubleLine
    .CloseAtPageBreak = True
    .HasDropShadow = True
    .Height = 5000
    .KeepTogether = True
    .Left = 750
    .LeftLineStyle = crLSDashLine
    .RightLineStyle = crLSDotLine
    .Suppress = False
    .Top = 100
    .TopLineStyle = crLSSingleLine
    .Width = 10000
End With

' Set the report source of the viewer and view the report.
CRViewer91.ReportSource = oReport
CRViewer91.ViewReport
End Sub

```

How to format an olap grid object

This example gets an olap grid object in the report header and then formats the olap grid object through the **OlapGridObject Object** properties. Import the **OLAP Cube Report.rpt** in the \Program Files\Crystal Decisions\Crystal Reports 9\Samples\En\Reports\Feature Examples folder into a Visual Basic application. See **“Open an existing report” on page 57** for information on importing reports.

```

' Create a new instance of the report.
Dim oReport As New CrystalReport1

Private Sub Form_Load()
    ' Declare an OlapGridObject object.
    Dim OlapGrd As CRAXDRT.OlapGridObject
    ' get the OlapGridObject object from the report header.
    Set OlapGrd = oReport.Sections("RH").ReportObjects(1)

    ' Format the olap grid object.

```

```

With OlapGrd
    .BackColor = vbGreen
    .BorderColor = vbRed
    .BottomLineStyle = crLSDoubleLine
    .TopLineStyle = crLSDashLine
    .RightLineStyle = crLSNoLine
    .LeftLineStyle = crLSSingleLine
    .CloseAtPageBreak = True
    .HasDropShadow = True
    .KeepTogether = True
    .Left = 300
    .Suppress = False
    .Top = 200
End With

' Set the report source of the viewer and view the report.
CRViewer91.ReportSource = oReport
CRViewer91.ViewReport
End Sub

```

How to format an OLE object

This example gets an OLE object and then formats the OLE object through the **OleObject Object** properties. CrystalReport1 is connected to the xtreme.mdb database found in the \Program Files\Crystal Decisions\Crystal Reports 9\Samples\En\Databases folder. The report is using the Customer table and an OLE is inserted into the report. See “Inserting OLE objects into reports” in the *Crystal Reports Online Help (crw.chm)*.

```

' Create a new instance of the report.
Dim oReport As New CrystalReport1

Private Sub Form_Load()
    ' Declare an OleObject object.
    Dim oOleObject As craxdrt.OleObject
    ' Declare a generic Object.
    Dim oObject As Object
    ' Declare a Section object.
    Dim oSection As craxdrt.Section
    ' This variable is set to true once the OLE object is found.
    Dim bOle As Boolean
    bOle = False

    ' Search for all the report objects in each section.
    For Each oSection In oReport.Sections
        For Each oObject In oSection.ReportObjects
            ' Find the first OLE object.
            If oObject.Kind = craxdrt.crOLEObject Then
                ' Get the graph object and exit the loop.
            End If
        Next oObject
    Next oSection

```

```

        Set oOleObject = oObject
        bOle = True
        Exit For
    End If
Next oObject
If bOle Then Exit For
Next oSection

' Format the blobfield object.
With oOleObject
    .BackColor = vbGreen
    .BorderColor = vbRed
    .TopLineStyle = crLSDoubleLine
    .LeftLineStyle = crLSDoubleLine
    .RightLineStyle = crLSDoubleLine
    .BottomLineStyle = crLSDoubleLine
    .HasDropShadow = True
    .BottomCropping = 1000
    .BottomLineStyle = crLSSingleLine
    .Left = 1000
    .Height = 10000
    .LeftCropping = 300
    .RightCropping = 300
    .RightLineStyle = crLSDashLine
    .Suppress = True
    .Top = 3000
    .TopCropping = 300
    .Width = 15000
    .XScaling = 5
    .YScaling = 1
    .Suppress = False
End With

' Set the report source of the viewer and view the report.
CRViewer91.ReportSource = oReport
CRViewer91.ViewReport
End Sub

```

How to format a subreport object

This example gets a subreport object and then formats the subreport object through the **SubreportObject Object** properties. CrystalReport1 is connected to the **xtrme.mdb** database found in the **\Program Files\Crystal Decisions\Crystal Reports 9\Samples\En\Databases** folder. The report is using the **Customer** table and a subreport using the **Details** table is added. See “Inserting subreports” in the *Crystal Reports Online Help (crw.chm)*.


```

' Create a new instance of the report.
Dim oReport As New CrystalReport1

Private Sub Form_Load()
' Declare a SubreportObject object.
Dim oSubreportObject As CRAXDRT.SubreportObject
' Declare a generic Object.
Dim oObject As Object
' Declare a Section object.
Dim oSection As CRAXDRT.Section
' This variable is set to true once the
' subreport object is found.
Dim bSubreport As Boolean
bSubreport = False

' Search for all the report objects in each section.
For Each oSection In oReport.Sections
    For Each oObject In oSection.ReportObjects
        ' Find the first subreport object.
        If oObject.Kind = crSubreportObject Then
            ' Get the subreport object and exit the loop.
            Set oSubreportObject = oObject
            bSubreport = True
            Exit For
        End If
    Next oObject
    If bSubreport Then Exit For
Next oSection

' Format the subreport object.
With oSubreportObject
    .BackColor = vbYellow
    .BorderColor = vbBlue
    .HasDropShadow = True
    .CanGrow = True
    .CloseAtPageBreak = True
    .KeepTogether = True
    .Height = 1200
    .Left = 2000
    .Top = 200
    .Width = 8000
    .BottomLineStyle = crLSDotLine
    .LeftLineStyle = crLSDashLine
    .RightLineStyle = crLSDoubleLine
    .TopLineStyle = crLSSingleLine
End With

' Set the report source of the viewer and view the report.
CRViewer91.ReportSource = oReport
CRViewer91.ViewReport
End Sub

```

How to format a text object

This example gets a text object and then formats the text object through the **TextObject Object** properties. CrystalReport1 is connected to the xtreme.mdb database found in the \Program Files\Crystal Decisions\Crystal Reports 9\Samples\En\Databases folder. The report is using the Customer table.

```
' Create a new instance of the report.
Dim oReport As New CrystalReport1

Private Sub Form_Load()
' Declare a TextObject object.
Dim oTextObject As CRAXDRT.TextObject
' Declare a generic Object.
Dim oObject As Object
' Declare a Section object.
Dim oSection As CRAXDRT.Section
' This variable is set to true once the
' text object is found.
Dim bText As Boolean
bText = False

' Search for all the report objects in each section.
For Each oSection In oReport.Sections
  For Each oObject In oSection.ReportObjects
    ' Find the first text object.
    If oObject.Kind = crTextObject Then
      ' Get the text object and exit the loop.
      Set oTextObject = oObject
      bText = True
      Exit For
    End If
  Next oObject
  If bText Then Exit For
Next oSection

' Format the text object and change the text.
With oTextObject
  .BackColor = vbYellow
  .BorderColor = vbRed
  .BottomLineStyle = crLSDoubleLine
  .CanGrow = True
  .CharacterSpacing = 125
  .CloseAtPageBreak = True
  .FirstLineIndent = 25
  .Font.Italic = True
  .Font.Bold = True
  .Font.Size = 14
  .HasDropShadow = True
  .Height = 900
  .HorizontalAlignment = crLeftAlign
  .KeepTogether = True
  .Left = 200
```

```

.LeftIndent = 750
.LeftLineStyle = crLSDashLine
.MaxNumberOfLines = 16
.RightIndent = 250
.RightLineStyle = crLSDotLine
.Suppress = False
.SuppressIfDuplicated = True
.SetText "Hello World"
.TextColor = vbBlue
.TextRotationAngle = crRotate0
.Top = 25
.TopLineStyle = crLSSingleLine
.Width = 3000
End With

' Set the report source of the viewer and view the report.
CRViewer91.ReportSource = oReport
CRViewer91.ViewReport
End Sub

```

How to import a subreport and link to a main report

This example imports a report as a subreport using the **ImportSubreport Method (Section Object)** and then links the reports using the **Add Method (SubreportLinks Collection)**. CrystalReport1 is connected to the xtreme.mdb database found in the \Program Files\Crystal Decisions\Crystal Reports 9\Samples\En\Databases folder. The report is using the Customer table. A second report is created off the same data source in Crystal Reports using the Orders table. The report is saved in the sample application directory as Orders.rpt.

Note: Some methods require licensing. See *License Manager Help (License.hlp)* for more information.

```

' Create a new instance of the main report.
Dim oReport As New CrystalReport1

' Declare the variables for adding the subreport
' and subreport links.
Dim oSubreport As CRAXDRT.Report
Dim oSection As CRAXDRT.Section
Dim oSubReportObject As CRAXDRT.SubreportObject
Dim oSubreportLinks As CRAXDRT.SubreportLinks
Dim oSubreportLink As CRAXDRT.SubreportLink
Dim oMainDBFieldObject As CRAXDRT.DatabaseFieldDefinition
Dim oSubDBFieldObject As CRAXDRT.DatabaseFieldDefinition

Private Sub Form_Load()
' Import a subreport into the main report Details "B"
' section. Get the "Db" section of the report.
Set oSection = oReport.Sections.Item("Db")
oSection.Suppress = False

```

```

' Import the subreport. The report is located in the
' application path.
' Note: The ImportSubreport method requires licensing.
Set oSubReportObject = oSection.ImportSubreport(App.Path & "\Orders.rpt", 0,
0)

' Open the subreport.
Set oSubReport = oSubReportObject.OpenSubreport

' Get the "CustomerID" field in the "Customer" table
' from the main report. This is the field that will be
' linked to the subreport.
Set oMainDBFieldObject = oReport.Database.Tables.Item(1).Fields.Item(1)

' Get the "CustomerID" field in the "Orders" table
' from the subreport. This is the field that will be
' linked to the main report.
Set oSubDBFieldObject = oSubReport.Database.Tables.Item(1).Fields.Item(3)

' Link the main report and the subreport.
Set oSubreportLinks = oSubReportObject.Links
Set oSubreportLink = oSubreportLinks.Add(oMainDBFieldObject,
oSubDBFieldObject)

' Set the report source of the viewer and view the report.
CRViewer91.ReportSource = oReport
CRViewer91.ViewReport
End Sub

```

How to re-import a subreport and link to a main report

This example links a subreport and a main report using the [Add Method \(SubreportLinks Collection\)](#) and then re-imports the subreport using the [ReImportSubreport Method \(SubreportObject Object\)](#). CrystalReport1 is connected to the xtreme.mdb database found in the \Program Files\Crystal Decisions\Crystal Reports 9\Samples\En\Databases folder. The report is using the Customer table. A second report is created off the same data source in Crystal Reports using the Orders table. The report is saved in the sample application directory as Orders.rpt. The report "Orders.rpt" is then imported into CrystalReport1 through the Designer's user interface at design time. See "Inserting subreports" in *Crystal Reports Online Help (crw.chm)* for more information.

Note: Some methods require licensing. See *License Manager Help (License.hlp)* for more information.

```

' Create a new instance of the main report.
Dim oReport As New CrystalReport1

' Declare the variables needed for adding
' the subreport links and reimporting the subreport.

```

```

Dim oSubreport As CRAXDRT.Report
Dim oSection As CRAXDRT.Section
Dim oReportObjects As CRAXDRT.ReportObjects
Dim oSubreportObject As CRAXDRT.SubreportObject
Dim oSubrpt As CRAXDRT.Report
Dim oReportObject As Object
Dim bSubreport As Boolean
Dim bReimported As Boolean

Private Sub Form_Load()
    ' Search for all the report objects in each section.
    For Each oSection In oReport.Sections
        Set oReportObjects = oSection.ReportObjects
        For Each oReportObject In oReportObjects
            If oReportObject.Kind = crSubreportObject Then
                Set oSubreportObject = oReportObject
                Set oSubrpt = oSubreportObject.OpenSubreport
                bSubreport = True
                Exit For
            End If
        Next
        If bSubreport Then Exit For
    Next

    ' Link the "Customer ID" field from the Customer
    ' table in the main report to the "Customer ID" field
    ' from the Orders table in the subreport.
    ' Note: The Links property requires licensing.
    oSubreportObject.Links.Add "{Customer.Customer ID}", "{Orders.Customer ID}"

    ' Import the subreport once again so that any changes
    ' are reflected in the main report.
    ' Note: The ReimportSubreport method requires licensing.
    oSubreportObject.ReimportSubreport bReimported

    ' If bReimported is True the subreport imported successfully.
    ' If bReimported is False the subreport failed to import.
    If bReimported Then
        MsgBox "Subreport imported successfully."
    Else
        MsgBox "Subreport failed to import."
        Exit Sub
    End If

    ' Set the report source of the viewer and view the report.
    CRViewer91.ReportSource = oReport
    CRViewer91.ViewReport
End Sub

```

How to add and delete table links

This example Demonstrates how to display information on a table link in a report using the properties of the **TableLink Object**, how to delete a table link using the **Delete Method (TableLinks Collection)**, and finally how to add a table link using the **Add Method (TableLinks Collection)**. CrystalReport1 is connected to the **xtreme.mdb** database found in the **\Program Files\Crystal Decisions\Crystal Reports 9\Samples\En\Databases** folder. The report is using the **Customer** table and the **Orders** table. The two tables are linked on the **Customer ID** field.

Note: Some methods require licensing. See *License Manager Help (License.hlp)* for more information.

```
' Create a new instance of the main report.
Dim oReport As New CrystalReport1

' Declare the variables needed for adding
' the table links.
Dim oTableLinks As CRAXDRT.TableLinks
Dim oTableLink As CRAXDRT.TableLink
Dim oSourceDBTable As CRAXDRT.DatabaseTable
Dim oDestinationDBTable As CRAXDRT.DatabaseTable
Dim oSourceDBField As CRAXDRT.DatabaseFieldDefinition
Dim oDestinationDBField As CRAXDRT.DatabaseFieldDefinition

Private Sub Form_Load()
' Get the collection of table links.
Set oTableLinks = oReport.Database.Links

' Display the information on the first table link.
MsgBox " The number of Table Links in this report is: " _
& oTableLinks.Count
' Get the first table link.
Set oTableLink = oTableLinks.Item(1)
MsgBox " The Source table for the link is: " _
& oTableLink.SourceTable.Name
MsgBox " The Source field for the link is: " _
& oTableLink.SourceFields(1).Name
MsgBox " The Destination table for the link is: " _
+ oTableLink.DestinationTable.Name
MsgBox " The Destination field for the link is: " _
+ oTableLink.DestinationFields.Item(1).Name
MsgBox " The TableLink index used is: " _
& oTableLink.IndexUsed
MsgBox " The TableLink join type is: " _
& oTableLink.JoinType
MsgBox " The TableLink lookup type is: " _
& oTableLink.LookupType
MsgBox " PartialMatchEnabled (True/False): " _
& oTableLink.PartialMatchEnabled
```

```

' Delete the table link.
' Note: The Delete method requires licensing.
oTableLinks.Delete (1)
MsgBox " Table link has been deleted "

' Get the tables and fields needed to create the link.
Set oSourceDBTable = oReport.Database.Tables(1)
Set oDestinationDBTable = oReport.Database.Tables(2)
Set oSourceDBField = oSourceDBTable.Fields.Item(1)
Set oDestinationDBField = oDestinationDBTable.Fields.Item(3)

' Link the "Customer ID" field of the Customer table
' to the "Customer ID" field of the Orders table.
' Note: The Add method requires licensing.
Set oTableLink = oTableLinks.Add(oSourceDBTable, oDestinationDBTable, _
                                oSourceDBField, oDestinationDBField, _
                                crJTEqual, crLTLookupParallel, False, True)

MsgBox " {Customer.CustomerID} has been linked to {Orders.CustomerID} "

' Set the report source of the viewer and view the report.
CRViewer91.ReportSource = oReport
CRViewer91.ViewReport
End Sub

```

How to add, delete and format groups in a crosstab

This example demonstrates how to delete a group from a crosstab using the **Delete Method (CrossTabGroups Collection)**, how to add a group to a crosstab using the **Add Method (CrossTabGroups Collection)**, and finally how to format the groups in a crosstab using the **CrossTabGroup Object** properties. The example consists of a form and a report designed in the Visual Basic IDE using the RDC. CrystalReport1 is connected to the xtreme.mdb database found in the \Program Files\Crystal Decisions\Crystal Reports 9\Samples\En\Databases folder. The report is using the Customer table.

Note: Some methods require licensing. See *License Manager Help (License.hlp)* for more information.

To use the example

- 1 Create a standard application.
- 2 Create a report using the RDC and the data source noted in the introduction.
 - Group the report on the Country field.
 - Add a crosstab to the group header. The crosstab uses the following fields:
 - Rows:
 - Customer.Customer Name

- Columns:
Customer.Country
Customer.Region
 - Summarized Fields:
Sum of Last Year's Sales
- 3 Add the following controls to Form1:
 - Button (Command1)
Command1 deletes a group from the crosstab and refreshes the viewer.
 - Button (Command2)
Command2 adds a group to the crosstab and refreshes the viewer.
 - Button (Command3)
Command3 formats the groups in the crosstab and refreshes the viewer.
 - 4 Cut and paste the code that is listed under Form1 below into the code window for Form1 in your application.
 - 5 Run the application.
 - 6 With the application running the user can:
 - Click Command1 to delete the Region column from the crosstab.
 - Click Command2 to add the Region column back to the crosstab.
 - Click Command3 to:
 - Change the reports grouping order to be specified by England, Canada and Germany.
 - Change the crosstabs grouping order to match the reports grouping order.
 - Format the Country group in the crosstab.

```
' Create a new instance of the main report.
Dim oReport As New CrystalReport1

' Declare the variables needed for adding
' deleting and formatting the crosstab groups.
Dim oCrossTabGroups As CRAXDRT.CrosstabGroups
Dim oCrossTabObject As CRAXDRT.CrosstabObject
Dim oDBFieldDefinition As CRAXDRT.DatabaseFieldDefinition
Dim oCrossTabGroup As CRAXDRT.CrosstabGroup
Dim varSpecifiedGroups(1, 2) As Variant
Dim sFieldName As String

Private Sub Form_Load()
' Get the crosstab object and crosstab groups collection
Set oCrossTabObject = oReport.Sections("GH").ReportObjects.Item(2)
Set oCrossTabGroups = oCrossTabObject.ColumnGroups

' Set the report source of the viewer and view the report.
CRViewer91.ReportSource = oReport
CRViewer91.ViewReport
End Sub
```



```

Private Sub Command1_Click()
    ' Delete the second group in the crosstab.
    If oCrossTabGroups.Count > 1 Then
        sFieldName = oCrossTabGroups.Item(2).Field.Name
        oCrossTabGroups.Delete (2)
        MsgBox "The CrossTab Group based on the field" & sFieldName & _
            " has been deleted."
    Else
        MsgBox "There is only one CrossTab group in this CrossTab Object." _
            + "This CrossTabGroup can NOT be deleted."
    End If
    ' Refresh the viewer.
    CRViewer91.Refresh
End Sub

Private Sub Command2_Click()
    ' Get the Region field and add it to the cross tab.
    Set oDBFieldDefinition = oReport.Database.Tables(1).Fields(12)
    oCrossTabGroups.Add oDBFieldDefinition
    MsgBox "A new CrossTabGroup Column Group based on the " & _
        & oDBFieldDefinition.Name & _
        " has been added to the CrossTab object."
    ' Refresh the viewer.
    CRViewer91.Refresh
End Sub

Private Sub Command3_Click()
    ' Create an array of headings and groups
    ' to use as specified groups in the report group
    ' and the crosstab group.

    ' Column headings.
    varSpecifiedGroups(0, 0) = "England"
    varSpecifiedGroups(0, 1) = "Canada"
    varSpecifiedGroups(0, 2) = "Germany"

    ' Column groups.
    varSpecifiedGroups(1, 0) = "{Customer.Country} = 'England'"
    varSpecifiedGroups(1, 1) = "{Customer.Country} = 'Canada'"
    varSpecifiedGroups(1, 2) = "{Customer.Country} = 'Germany'"

    Set oCrossTabGroup = oCrossTabGroups.Item(1)

    ' Set the group to use a specified order and groups.
    oReport.Areas("GH").SortDirection = crSpecifiedOrder
    oReport.Areas("GH").SpecifiedGroups = varSpecifiedGroups

    ' Format the crosstab.
    With oCrossTabGroup
        .BackColor = vbGreen
        .Condition = crGCAnyValue
        .EnableSuppressLabel = False
        .EnableSuppressSubtotal = False
    End With

```

```

.SortDirection = crSpecifiedOrder
' Set the crosstab group to use a specified order and groups.
.SortDirection = crSpecifiedOrder
.SpecifiedGroups = varSpecifiedGroups

MsgBox "The CrossTab Group that is being formatted is based on the " & _
    .Field.Name & " field."
End With

' Refresh the viewer.
CRViewer91.Refresh
End Sub

```

How to format a group in the section format event

This example changes the color of the text displayed in the group header based on the value of the group. The value is checked through the Value and NextValue properties of the **GroupNameFieldDefinition Object**. The values are checked in the **Format Event (Section Object)**. CrystalReport1 is connected to the xtreme.mdb database found in the \Program Files\Crystal Decisions\Crystal Reports 9\Samples\En\Databases folder. The report is using the Customer table and is grouped on the Country field and then the Last Year's Sales field.

```

' Create a new instance of the report.
Dim oReport As New CrystalReport1
' Declare a GroupNameFieldDefinitions collection.
Dim cGroupNameFieldDefinitions As CRAXDRT.GroupNameFieldDefinitions
' Declare a GroupNameFieldDefinition object.
Dim oGroupNameFieldDefinition As CRAXDRT.GroupNameFieldDefinition

' Declare a Section object with events.
Dim WithEvents oSection As CRAXDRT.Section

Dim x As Integer

Private Sub Form_Load()
' Set the section to the Last Year's sales group header.
Set oSection = oReport.Sections("GH2")
' Get the collection of group name fields.
Set cGroupNameFieldDefinitions = oReport.GroupNameFields

' Set the report source of the viewer and view the report.
CRViewer91.ReportSource = oReport
CRViewer91.ViewReport
End Sub

' Note: You can also place code for a section format
' event in a Report form (for example CrystalReport1) found
' under the Designer folder. Double-click the desired section

```

```

' and place the code in the code window. If your report
' is a file (*.rpt) then you must declare a section
' object with events and select the section object and then
' format in the forms code window.
Private Sub oSection_format(ByVal pFormattingInfo As Object)

' While the section is formatting check the value of the
' GroupNameFieldDefinition object. Change the text color
' based on that value.
If cGroupNameFieldDefinitions.Item(2).Value > 20000 Then
    oReport.Sections("GH2").ReportObjects.Item(1).TextColor = vbGreen
Else
    ' If the value is less than 20000 compare the next value with the
    ' present value to determine the text color.
    If cGroupNameFieldDefinitions.Item(2).NextValue >
cGroupNameFieldDefinitions.Item(2).Value Then
        oReport.Sections("GH2").ReportObjects.Item(1).TextColor = vbBlue
    Else: oReport.Sections("GH2").ReportObjects.Item(1).TextColor = vbRed
    End If
End If
End Sub

```

Enumerated Types

CRAAlignment

Constant	Value
crDefaultAlign	0
crHorCenterAlign	2
crJustified	4
crLeftAlign	1
crRightAlign	3

CRAMPMTYPE

Constant	Value
crAmPmAfter	1
crAmPmBefore	0

CRAreaKind

Constant	Value
crDetail	4
crGroupFooter	5
crGroupHeader	3
crPageFooter	7
crPageHeader	2
crReportFooter	8
crReportHeader	1

CRBarSize

Constant (numeric order)	Description
crMinimumBarSize	0
crSmallBarSize	1
crAverageBarSize	2
crLargeBarSize	3
crMaximumBarSize	4

CRBindingMatchType

Constant	Value
crBMTName	0
crBMTNameAndValue	1

CRBooleanFieldFormatConditionFormulaType

Constant	Value
crOutputFormatConditionFormulaType	20 (&H14)

CRBooleanOutputType

Constant	Value
crOneOrZero	4
crTOrF	1
crTrueOrFalse	0
crYesOrNo	2
crYOrN	3

CRBorderConditionFormulaType

Constant	Value
crBackgroundColorConditionFormulaType	45 (&H2D)
crBorderColorConditionFormulaType	46 (&H2E)
crBottomLineStyleConditionFormulaType	43 (&H2B)
crHasDropShadowConditionFormulaType	44 (&H2C)
crLeftLineStyleConditionFormulaType	40 (&H28)
crRightLineStyleConditionFormulaType	41 (&H29)
crTightHorizontalConditionFormulaType	47 (&H2F)
crTightVerticalConditionFormulaType	48 (&H30)
crTopLineStyleConditionFormulaType	42 (&H2A)

CRCommonFieldFormatConditionFormulaType

Constant	Value
crSuppressIfDuplicatedConditionFormulaType	80
crUseSystemDefaultConditionFormulaType	81 (&H51)

CRConvertDateTimeType

Constant	Value
crConvertDateTimeToDate	1
crConvertDateTimeToString	0
crKeepDateTimeType	2

CRCurrencyPositionType

Constant	Value
crLeadingCurrencyInsideNegative	0
crLeadingCurrencyOutsideNegative	1
crTrailingCurrencyInsideNegative	2
crTrailingCurrencyOutsideNegative	3

CRCurrencySymbolType

Constant	Value
crCSTFixedSymbol	1
crCSTFloatingSymbol	2
crCSTNoSymbol	0

CRDatabaseType

Constant	Value
crSQLDatabase	2
crStandardDatabase	1

CRDateCalendarType

Constant	Value
crGregorianCalendar	1
crGregorianUSCalendar	2
crGregorianArabicCalendar	10
crGregorianMEFrenchCalendar	9
crGregorianXlitEnglishCalendar	11
crGregorianXlitFrenchCalendar	12
crHebrewCalendar	8
crHijriCalendar	6
crJapaneseCalendar	3
crKoreanCalendar	5
crTaiwaneseCalendar	4
crThaiCalendar	7

CRDateEraType

Constant	Value
crLongEra	1
crNoEra	2
crShortEra	0

CRDateFieldFormatConditionFormulaType

Constant	Value
crCalendarTypeConditionFormulaType	131 (&H83)
crDateFirstSeparatorConditionFormulaType	128 (&H80)
crDateOrderConditionFormulaType	124 (&H7C)
crDatePrefixSeparatorConditionFormulaType	132 (&H84)
crDateSecondSeparatorConditionFormulaType	129 (&H81)
crDateSuffixSeparatorConditionFormulaType	133 (&H85)
crDayFormatConditionFormulaType	122 (&H7A)
crDayOfWeekEnclosureConditionFormulaType	134 (&H86)
crDayOfWeekPositionConditionFormulaType	127 (&H7F)
crDayOfWeekSeparatorConditionFormulaType	126 (&H7E)
crDayOfWeekTypeConditionFormulaType	125 (&H7D)
crEraTypeConditionFormulaType	130 (&H82)
crMonthFormatConditionFormulaType	121 (&H79)
crWindowsDefaultTypeConditionFormulaType	123 (&H7B)
crYearFormatConditionFormulaType	120 (&H78)

CRDateOrder

Constant	Value
crDayMonthYear	1
crMonthDayYear	2
crYearMonthDay	0

CRDateTimeFieldFormatConditionFormulaType

Constant	Value
crDateTimeOrderConditionFormulaType	140 (&H8C)
crDateTimeSeparatorConditionFormulaType	141 (&H8D)

CRDateWindowsDefaultType

Constant	Value
crNotUsingWindowsDefaults	2
crUseWindowsLongDate	0
crUseWindowsShortDate	1

CRDayType

Constant	Value
crLeadingZeroNumericDay	1
crNoDay	2
crNumericDay	0

CRDiscreteOrRangeKind

Constant (numeric order)	Value
crDiscreteValue	0
crRangeValue	1
crDiscreteAndRangeValue	2

CRDivisionMethod

Constant	Value
crAutomaticDivision	0
crManualDivision	1

CRExchangeDestinationType

Constant	Value
crExchangeFolderType	0
crExchangePostDocMessage	1011 (&H3F3)

CRExportDestinationType

Constant	Value
crEDTApplication	5
crEDTDiskFile	1
crEDTEMailMAPI	2
crEDTEMailVIM	3
crEDTLotusDomino	6
crEDTMicrosoftExchange	4
crEDTNoDestination	0

CRExportFormatType

Constant	Value
crEFTCharSeparatedValues	7
crEFTCommaSeparatedValues	5
crEFTCrystalReport	1
crEFTCrystalReport70	33
crEFTDataInterchange	2
crEFTEExactRichText	35 (&H23)
crEFTEExcel50	21 (&H15)
crEFTEExcel50Tabular	22 (&H16)
crEFTEExcel70	27 (&H1B)
crEFTEExcel70Tabular	28 (&H1C)
crEFTEExcel80	29 (&H1D)
crEFTEExcel80Tabular	30 (&H1E)
crEFTEExcel97	36 (&H24)
crEFTEExplorer32Extend	25 (&H19)
crEFTHTML32Standard	24 (&H18)

Constant	Value
crEFTHTML40	32 (&H20)
crEFTLotus123WK1	12
crEFTLotus123WK3	13
crEFTLotus123WKS	11
crEFTNoFormat	0
crEFTODBC	23 (&H17)
crEFTPaginatedText	10
crEFTPortableDocFormat	31 (&H1F)
crEFTRecordStyle	3
crEFTReportDefinition	34 (&H22)
crEFTTabSeparatedText	9
crEFTTabSeparatedValues	6
crEFTText	8
crEFTWordForWindows	14
crEFTXML	37 (&H25)

CRFieldKind

Constant	Value
crDatabaseField	1
crFormulaField	2
crGroupNameField	5
crRunningTotalField	7
crParameterField	6
crSpecialVarField	4
crSQLExpressionField	8
crSummaryField	3

CRFieldMappingType

Constant	Value
crAutoFieldMapping	0
crEventFieldMapping	2
crPromptFieldMapping	1

CRFieldValueType

Constant	Value
crBitmapField	17 (&H11)
crBlobField	15
crBooleanField	9
crChartField	21 (&H15)
crCurrencyField	8
crDateField	10
crDateTimeField	16 (&H10)
crIconField	18 (&H12)
crInt16sField	3
crInt16uField	4
crInt32sField	5
crInt32uField	6
crInt8sField	1
crInt8uField	2
crNumberField	7
crOleField	20 (&H14)
crPersistentMemoField	14
crPictureField	19 (&H13)
crStringField	12
crTimeField	11
crTransientMemoField	13
crUnknownField	22 (&H16)

CRFontColorConditionFormulaType

Constant	Value
crColorConditionFormulaType	180 (&HB4)
crFontConditionFormulaType	181 (&HB5)
crFontSizeConditionFormulaType	183 (&HB7)
crFontStrikeOutConditionFormulaType	184 (&HB8)
crFontStyleConditionFormulaType	182 (&HB6)
crFontUnderLineConditionFormulaType	185 (&HB9)

CRFormulaSyntax

Constant	Value
crBasicSyntaxFormula	1
crCrystalSyntaxFormula	0 Default value

CRGraphColor

Constant	Value
crBlackAndWhiteGraph	1
crColorGraph	0

CRGraphDataPoint

Constant	Value
crNone	0
crShowLabel	1
crShowValue	2

CRGraphDataType

Constant	Value
crCrossTabGraph	2
crDetailGraph	1
crGroupGraph	0

CRGraphDirection

Constant	Value
crHorizontalGraph	0
crVerticalGraph	1

CRGraphType

Constant	Value
crAbsoluteAreaGraph	20 Obsolete
crDualAxisBubbleGraph	91 Obsolete
crFaked3DAbsoluteAreaGraph	23 Obsolete
crFaked3DPercentAreaGraph	25 (&H19)
crFaked3DPercentBarGraph	5
crFaked3DRegularPieGraph	31 (&H1F)
crFaked3DSideBySideBarGraph	3
crFaked3DStackedAreaGraph	24 (&H18)
crFaked3DStackedBarGraph	4
crHighLowDualAxisGraph	101 Obsolete.
crHighLowGraph	100 (&H64)
crHighLowOpenCloseDualAxisGraph	105 Obsolete.
crHighLowOpenCloseGraph	104 (&H68)
crHighLowOpenDualAxisGraph	103 Obsolete.
crHighLowOpenGraph	102 Obsolete.
crLineGraphWithMarkers	13
crMultipleDoughnutGraph	41 (&H29)
crMultiplePieGraph	32 (&H20)
crMultipleProportionalDoughnutGraph	42 (&H2A)
crMultipleProportionalPieGraph	33 (&H21)
crPercentageLineGraph	12
crPercentageLineGraphWithMarkers	15
crPercentAreaGraph	22 (&H16)
crPercentBarGraph	2
crRadarDualAxisGraph	82 Obsolete
crRegularBubbleGraph	90 (&H5A)
crRegularDoughnutGraph	40 (&H28)
crRegularLineGraph	10
crRegularPieGraph	30 (&H1E)
crRegularRadarGraph	80 (&H50)
crSideBySideBarGraph	0
crStackedAreaGraph	21 (&H15)
crStackedBarGraph	1

Constant	Value
crStackedLineGraph	11
crStackedLineGraphWithMarkers	14
crStackedRadarGraph	81 (&H51)
crThreeDCutCornersGraph	53 (&H35)
crThreeDOctagonGraph	52 (&H34)
crThreeDPyramidGraph	51 (&H33)
crThreeDRegularGraph	50 (&H32)
crThreeDSurfaceHoneycombGraph	62 (&H3E)
crThreeDSurfaceRegularGraph	60 (&H3C)
crThreeDSurfaceWithSidesGraph	61 (&H3D)
crUnknownGraph	1000 (&H3E8)
crXyScatterGraph	70 (&H46)

CRGridlineType

Constant	Value
crMajorAndMinorGridlines	3
crMajorGridlines	2
crMinorGridlines	1
crNoGridlines	0

CRGroupCondition

Constant	Value
crGCAnnually	7
crGCAnyValue	14
crGCBiweekly	2
crGCByAMPM	18 (&H12)
crGCByHour	17 (&H11)
crGCByMinute	16 (&H10)
crGCBySecond	15
crGCDaily	0
crGCEveryNo	11
crGCEveryYes	10

Constant	Value
crGCMonthly	4
crGCNextIsNo	13
crGCNextIsYes	12
crGCQuarterly	5
crGCSEmiAnnually	6
crGCSEmiMonthly	3
crGCToNo	9
crGCToNo	8
crGCWeekly	1

CRHierarchicalSummaryType

Constant	Value
crHierarchicalSummaryNone	0
crSummaryAcrossHierarchy	1

CRHourType

Constant	Value
crNoHour	2
crNumericHour	0
crNumericHourNoLeadingZero	1

CRHTMLPageStyle

Constant	Value
crFramePageStyle	2
crPlainPageStyle	0
crToolbarAtBottomPageStyle	4
crToolbarAtTopPageStyle	3
crToolbarPageStyle	1

CRHTMLToolbarStyle

These bitwise constants can be XOR'd to specify the toolbar style.

Constant	Value
crToolbarRefreshButton	1
crToolbarSearchBox	2

CRImageType

Constant	Value
crDIBImageType	1
crJPEGImageType	2
crImageUnknown	0

CRLeadingDayPosition

Constant	Value
crLeadingDayOfWeek	0
crTrailingDayOfWeek	1

CRLeadingDayType

Constant	Value
crLongLeadingDay	1
crNoLeadingDay	2
crShortLeadingDay	0

CRLegendPosition

Constant	Value
crPlaceLeft	5
crPlaceRight	4
crPlaceBottom	6
crPlaceCustom	7

CRLinespacingType

Constant	Value
crExactSpacing	1
crMultipleSpacing	0

CRLineStyle

Constant	Value
crLSDashLine	3
crLSDotLine	4
crLSDoubleLine	2. Not valid for LineObject.LineStyle and BoxObject.LineStyle.
crLSNoLine	0. Not valid for LineObject.LineStyle and BoxObject.LineStyle.
crLSSingleLine	1

CRLinkJoinType

Constant	Value
crJTAdvance	13
crJTEqual	4
crJTGreaterOrEqual	10
crJTGreaterThan	8
crJTLeftOuter	5
crJTLessOrEqual	11
crJTLessThan	9
crJTNotEqual	12
crJTRightOuter	6

CRLinkLookupType

Constant	Value
crLTLookupParallel	1
crLTLookupProduct	2
crLTLookupSeries	3

CRMarkerShape

Constant	Value
crCircleShape	4
crDiamondShape	5
crRectangleShape	1
crTriangleShape	8

CRMarkerSize

Constant	Value
crLargeMarkers	4
crMediumLargeMarkers	3
crMediumMarkers	2
crMediumSmallMarkers	1
crSmallMarkers	0

CRMinuteType

Constant	Value
crNoMinute	2
crNumericMinute	0
crNumericMinuteNoLeadingZero	1

CRMonthType

Constant	Value
crLeadingZeroNumericMonth	1
crLongMonth	3
crNoMonth	4
crNumericMonth	0
crShortMonth	2

CRNegativeType

Constant	Value
crBracketed	3
crLeadingMinus	1
crNotNegative	0
crTrailingMinus	2

CRNumberFormat

Constant	Value
crCurrencyMillions	12
crCurrencyNoDecimal	3
crCurrencyThousands	11
crCurrencyTwoDecimal	4
crCustomNumberFormat	8
crMillionsNoDecimal	10
crNoDecimal	0
crOneDecimal	1
crPercentNoDecimal	5
crPercentOneDecimal	6
crPercentTwoDecimal	7
crThousandsNoDecimal	9
crTwoDecimal	2

CRNumericFieldFormatConditionFormulaType

Constant	Value
crAllowFieldClippingConditionFormulaType	114 (&H72)
crCurrencyPositionConditionFormulaType	111 (&H6F)
crCurrencySymbolConditionFormulaType	109 (&H6D)
crCurrencySymbolFormatConditionFormulaType	104 (&H68)
crDecimalSymbolConditionFormulaType	108 (&H6C)
crDisplayReverseSignConditionFormulaType	112 (&H70)
crEnableSuppressIfZeroConditionFormulaType	105 (&H69)
crEnableUseLeadZeroConditionFormulaType	102 (&H66)

Constant	Value
crHasOneSymbolPerPageConditionFormulaType	110 (&H6E))
crNDecimalPlacesConditionFormulaType	100 (&H64)
crNegativeFormatConditionFormulaType	103 (&H67)
crRoundingFormatConditionFormulaType	101 (&H65)
crThousandsSeparatorFormatConditionFormulaType	106 (&H6A)
crThousandSymbolFormatConditionFormulaType	107 (&H6B)
crZeroValueStringConditionFormulaType	113 (&H71)

CObjectFormatConditionFormulaType

Constant	Value
crCssClassConditionFormulaType	66 (&H42)
crEnableCanGrowConditionFormulaType	64 (&H40)
crEnableCloseAtPageBreakConditionFormulaType	62 (&H3E)
crEnableKeepTogetherConditionFormulaType	61 (&H3D)
crEnableSuppressConditinFormulaType	60 (&H3C)
crHorizontalAlignmentConditionFormulaType	63 (&H3F)
crHyperLinkConditionFormulaType	68 (&H44)
crRotationConditionFormulaType	67 (&H43)
crToolTipTextConditionFormulaType	65 (&H41)

CObjectKind

Constant	Value
crBlobFieldObject	9
crBoxObject	4
crCrossTabObject	8
crFieldObject	1
crGraphObject	7
crLineObject	3
crMapObject	10
crOlapGridObject	11
crOleObject	6
crSubreportObject	5
crTextObject	2

CROpenReportMethod

Constant	Value
crOpenReportByDefault	0
crOpenReportByTempCopy	1

CRPaperOrientation

Constant	Value
crDefaultPaperOrientation	0
crLandscape	2
crPortrait	1

CRPaperSize

Constant	Value
crDefaultPaperSize	0
crPaper10x14	16 (&H10)
crPaper11x17	17 (&H11)
crPaperA3	8
crPaperA4	9
crPaperA4Small	10
crPaperA5	11
crPaperB4	12
crPaperB5	13
crPaperCsheet	24 (&H18)
crPaperDsheet	25 (&H19)
crPaperEnvelope10	20 (&H14)
crPaperEnvelope11	21 (&H15)
crPaperEnvelope12	22 (&H16)
crPaperEnvelope14	23 (&H17)
crPaperEnvelope9	19 (&H13)
crPaperEnvelopeB4	33 (&H21)
crPaperEnvelopeB5	34 (&H22)
crPaperEnvelopeB6	35 (&H23)

Constant	Value
crPaperEnvelopeC3	29 (&H1D)
crPaperEnvelopeC4	30 (&H1E)
crPaperEnvelopeC5	28 (&H1C)
crPaperEnvelopeC6	31 (&H1F)
crPaperEnvelopeC65	32 (&H20)
crPaperEnvelopeDL	27 (&H1B)
crPaperEnvelopeItaly	36 (&H24)
crPaperEnvelopeMonarch	37 (&H25)
crPaperEnvelopePersonal	38 (&H26)
crPaperEsheet	26 (&H1A)
crPaperExecutive	7
crPaperFanfoldLegalGerman	41 (&H29)
crPaperFanfoldStdGerman	40 (&H28)
crPaperFanfoldUS	39 (&H27)
crPaperFolio	14
crPaperLedger	4
crPaperLegal	5
crPaperLetter	1
crPaperLetterSmall	2
crPaperNote	18 (&H12)
crPaperQuarto	15
crPaperStatement	6
crPaperTabloid	3
crPaperUser	256 (&H100)

CRPaperSource

Constant	Value
crPRBinAuto	7
crPRBinCassette	14
crPRBinEnvelope	5
crPRBinEnvManual	6
crPRBinFormSource	15
crPRBinLargeCapacity	11

crPRBinLargeFmt	10
crPRBinLower	2
crPRBinManual	4
crPRBinMiddle	3
crPRBinSmallFmt	9
crPRBinTractor	8
crPRBinUpper	1

CRParameterFieldType

Constant	Value
crQueryParameter	1
crReportParameter	0
crStoreProcedureParameter	2

CRParameterPickListSortMethod

Constant (numeric order)	Value
crNoSort	0
crAlphanumericAscending	1
crAlphanumericDescending	2
crNumericAscending	3
crNumericDescending	4

CRPieLegendLayout

Constant	Value
crAmountLayout	1
crBothLayout	2
crNoneLayout	3
crPercentLayout	0

CRPieSize

Constant (numeric order)	Value
crMaximumPieSize	0
crLargePieSize	16 (&H10)
crAveragePieSize	32 (&H20)
crSmallPieSize	48 (&H40)
crMinimumPieSize	64 (&H30)

CRPlaceHolderType

Constant	Value
crAllowPlaceHolders	2
crDelayTotalPageCountCalc	1

CRPrinterDuplexType

Constant	Value
crPRDPDefault	0
crPRDPHorizontal	3
crPRDPSimplex	1
crPRDPVertical	2

CRPrintingProgress

Constant	Value
crPrintingCancelled	5
crPrintingCompleted	3
crPrintingFailed	4
crPrintingHalted	6
crPrintingInProgress	2
crPrintingNotStarted	1

CRRangeInfo

Constant (numeric order)	Value
crRangeNotIncludeUpperLowerBound	0
crRangeIncludeUpperBound	1
crRangeIncludeLowerBound	2
crRangeNoUpperBound	4
crRangeNoLowerBound	8

CRRenderResultType

Constant	Value
crBSTRType	8. This constant is currently not supported.
crUISafeArrayType	8209

CRReportFileFormat

Constant	Value
cr70FileFormat	1792
cr80FileFormat	2048

CRReportKind

Constant	Value
crColumnarReport	1
crLabelReport	2
crMulColumnReport	3

CRReportVariableValueType

Constant	Value
crRVBoolean	2
crRVCurrency	1
crRVDate	3

Constant	Value
crRVDateTime	5
crRVNumber	0
crRVString	6
crRVTime	4

CRRotationAngle

Constant (numeric order)	Value
crRotate0	0
crRotate90	1
crRotate270	2

CRRoundingType

Constant (numeric order)	Value
crRoundToMillion	17
crRoundToHundredThousand	16
crRoundToTenThousand	15
crRoundToThousand	14
crRoundToHundred	13
crRoundToTen	12
crRoundToUnit	11
crRoundToTenth	10
crRoundToHundredth	9
crRoundToThousandth	8
crRoundToTenThousandth	7
crRoundToHundredThousandth	6
crRoundToMillionth	5
crRoundToTenMillionth	4
crRoundToHundredMillionth	3
crRoundToBillionth	2
crRoundToTenBillionth	1

CRRunningTotalCondition

Constant	Value
crRTEvalNoCondition	0
crRTEvalOnChangeOfField	1
crRTEvalOnChangeOfGroup	2
crRTEvalOnFormula	3

CRSearchDirection

Constant (numeric order)	Value
crForward	0
crBackward	1

CRSecondType

Constant	Value
crNumericNoSecond	2
crNumericSecond	0
crNumericSecondNoLeadingZero	1

CRSectionAreaFormatConditionFormulaType

Constant	Value
crSectionAreaBackgroundColorConditionFormulaType	9
crSectionAreaCssClassConditionFormulaType	8
crSectionAreaEnableHideForDrillDownConditionFormulaType	11
crSectionAreaEnableKeepTogetherConditionFormulaType	4
crSectionAreaEnableNewPageAfterConditionFormulaType	2
crSectionAreaEnableNewPageBeforeConditionFormulaType	3
crSectionAreaEnablePrintAtBottomOfPageConditionFormulaType	1
crSectionAreaEnableResetPageNumberAfterConditionFormulaType	6
crSectionAreaEnableSuppressConditionFormulaType	0
crSectionAreaEnableSuppressIfBlankConditionFormulaType	5
crSectionAreaEnableUnderlaySectionConditionFormulaType	7
crSectionAreaShowAreaConditionFormulaType	10

CRSliceDetachment

Constant (numeric order)	Value
crLargestSlice	2
crSmallestSlice	1
crNoDetachment	0

CRSortDirection

Constant	Value
crAscendingOrder	0
crDescendingOrder	1
crOriginalOrder	2. Not supported for any kind of groups.
crSpecifiedOrder	3. Not supported for any kind of groups.

CRSpecialVarType

Constant	Value
crSVTDataDate	4
crSVTDataTime	5
crSVTFileAuthor	15
crSVTFileCreationDate	16 (&H10)
crSVTFilename	14
crSVTGroupNumber	8
crSVTGroupSelection	13
crSVTModificationDate	2
crSVTModificationTime	3
crSVTPageNofM	17 (&H11)
crSVTPageNumber	7
crSVTPrintDate	0
crSVTPrintTime	1
crSVTRecordNumber	6
crSVTRecordSelection	12
crSVTReportComments	11
crSVTReportTitle	10
crSVTTotalPageCount	9

CRStringFieldConditionFormulaType

Constant	Value
crTextInterpretationConditionFormulaType	200 (&HC8)

CRSubreportConditionFormulaType

Constant	Value
crCaptionConditionFormulaType	220 (&HDC)
crDrillDownTabTextConditionFormulaType	221 (&HDD)

CRSummaryType

Constant	Value
crSTAverage	1
crSTCount	6
crSTDCorrelation	10
crSTDCovariance	11
crSTDDistinctCount	9
crSTDMedian	13
crSTDMode	17 (&H11)
crSTDNthLargest	15
crSTDNthMostFrequent	18 (&H12)
crSTDNthSmallest	16 (&H10)
crSTDPercentage	19 (&H13)
crSTDPercentile	14
crSTDWeightedAvg	12
crSTMaximum	4
crSTMinimum	5
crSTPopStandardDeviation	8
crSTPopVariance	7
crSTSsampleStandardDeviation	3
crSTSsampleVariance	2
crSTSum	0

CRTableDifferences

Constant	Value
crTDOK	0x00000000
crTDDatabaseNotFound	0x00000001
crTDServerNotFound	0x00000002
crTDServerNotOpened	0x00000004
crTDAliasChanged	0x00000008
crTDIndexesChanged	0x00000010
crTDDriverChanged	0x00000020
crTDDictionaryChanged	0x00000040
crTDFileTypeChanged	0x00000080
crTDRecordSizeChanged	0x00000100
crTDAccessChanged	0x00000200
crTDParametersChanged	0x00000400
crTDLocationChanged	0x00000800
crTDDatabaseOtherChanges	0x00001000
crTDNumberFieldChanged	0x00010000
crTDFieldOtherChanges	0x00020000
crTDFieldNameChanged	0x00040000
crTDFieldDescChanged	0x00080000
crTDFieldTypeChanged	0x00100000
crTDFieldSizeChanged	0x00200000
crTDNativeFieldTypeChanged	0x00400000
crTDNativeFieldOffsetChanged	0x00800000
crTDNativeFieldSizeChanged	0x01000000
crTDFieldDecimalPlacesChanged	0x02000000

CRTextFormat

Constant	Value
crHTMLText	2
crRTFText	1
crStandardText	0

CRTIMEBase

Constant	Value
cr12Hour	0
cr24Hour	1

CRTIMEFieldFormatConditionFormulaType

Constant	Value
crAMPMFormatConditionFormulaType	161 (&HA1)
crAMStringConditionFormulaType	166 (&HA6)
crHourFormatConditionFormulaType	162 (&HA2)
crHourMinuteSeparatorConditionFormulaType	168 (&HA8)
crMinuteFormatConditionFormulaType	163 (&HA3)
crMinuteSecondSeparatorConditionFormulaType	167 (&HA7)
crPMStringConditionFormulaType	165 (&HA5)
crSecondFormatConditionFormulaType	164 (&HA4)
crTimeBaseConditionFormulaType	160 (&HA0)

CRTOPOrBottomNGroupSortOrder

Constant	Value
crAllGroupsSorted	1
crAllGroupsUnsorted	0
crBottomNGroups	3
crTopNGroups	2
crUnknownGroupsSortOrder	10

CRValueFormatType

Constant	Value
crAllowComplexFieldFormatting	4
crIncludeFieldValues	1
crIncludeHiddenFields	2

CRViewingAngle

Constant	Value
crBirdsEyeView	15
crDistortedStdView	10
crDistortedView	4
crFewGroupsView	9
crFewSeriesView	8
crGroupEmphasisView	7
crGroupEyeView	6
crMaxView	16 (&H10)
crShorterView	12
crShortView	5
crStandardView	1
crTallView	2
crThickGroupsView	11
crThickSeriesView	13
crThickStdView	14
crTopView	3

CRYearType

Constant	Value
crLongYear	1
crNoYear	2
crShortYear	0

Programming the Crystal Report Viewers **4**

The Crystal Report Viewer is a front-end user interface for viewing reports. In this chapter you will find detailed information on implementing the ActiveX and Java Bean viewers in your application.

Enhancements to the Report Viewer

The Report Viewer has been enhanced substantially:

- The Report Viewer uses multi-threading. As a result, your users can begin viewing a report sooner, even if the report requires that it all be run before certain values are generated (page numbering, for example, of the style “page 24 of 125”). In such a case, the Report Engine uses place holders for the yet-to-be-generated total page count. When that page count is completed, the Report Engine inserts the missing data into the pages already read.
- The report’s group tree is loaded on-demand. This allows your users to use the tree functionality for navigation even when only a partial group tree has been loaded.
- You can specify a page number to go to in the report you are currently viewing.
- You can use the Select Expert and the Search Expert in the viewer to select records and search for specific values using formulas.
- The Report Viewer supports the use of rotated text in the report.
- The toolbar for the Report Viewer for ActiveX has a new look.
- You can customize the Report Viewer by resizing sections of the toolbar, adding custom bitmaps, and more.
- There is a Help button implemented for applications. Clicking on the Help button can fire an event to your application so it can display the appropriate help.
- There are over 30 events giving you the ability to make previewing the report a truly interactive activity.

For a better understanding of all the capabilities of the Report Viewer, review the viewer object model (CRVIEWERLibCtl) in the Visual Basic Object Browser.

Note: Visit the Developer Zone web site at:

http://www.crystaldecisions.com/products/dev_zone

Click Support to get links for finding documentation and knowledge base articles about the Report Designer Component.

Application Development with Crystal Report Viewers

Developing applications that display reports on screen is now a straightforward process. Crystal Reports includes the Crystal Report Viewers as easy to use but complex components that can be embedded directly in an application. Once added to an application, reports accessed through the Report Engine Automation Server, the Report Designer Component, or the Crystal Web Reports Server can be displayed right inside your own applications. The Report Viewer retains all of the powerful formatting, grouping, and totalling power of the original report, and your users get access to data in a dynamic and clear format.

Crystal Reports provides two Report Viewers specifically designed for application development: the Crystal Report Viewer for ActiveX and the Crystal Report Viewer Java Bean. Both provide a complete object model for programming and manipulating the Report Viewer at runtime inside your applications. Simply displaying a single report inside the Report Viewer is a simple process requiring only a couple of lines of code. However, if necessary for your application, you have the option of complete control over how the Report Viewer appears and functions.

With the Crystal Report Viewer as a front-end user interface for viewing reports, Crystal Reports development technologies allow you to develop even complex client/server or multi-tier applications that access, manipulate, and display data for intranet systems, workgroups, or any group of people needing clear and informative data on a regular basis. Design robust Business Support systems and Enterprise Information Management applications delivering even the most complex data through the Crystal Report Viewers.

This chapter describes both the ActiveX and Java Bean versions of the Report Viewer in relation to designing applications using Crystal Reports development technologies.

Crystal Report Viewer for ActiveX

The Crystal Report Viewer for ActiveX is a standard ActiveX control that can be added to an application in any development environment that supports ActiveX. Programmers using Visual Basic, Delphi, Visual C++, or Borland C++ programmers all receive the benefit of quickly adding a powerful report viewer to an application with little coding.

As a standard component, the ActiveX Report Viewer exposes several properties at design time, but also provides a complete object model with properties, methods, and events that can be programmed at runtime. The following sections discuss various topics for working with the ActiveX Report Viewer in Visual Basic. If you are using a development environment other than Visual Basic, use these topics as a guideline, but refer to your development software documentation for specific information on working with ActiveX controls.

The Crystal Report Viewer, as an ActiveX control, includes a complete object model for controlling how it appears in an application, and how it displays reports. Simply displaying a report in the Report Viewer window takes little code, but to truly make use of its power requires a broader understanding of how to work with the object model.

Related topics:

[“Adding the Report Viewer to a Visual Basic project” on page 234](#)

[“Using the CRViewer object” on page 234](#)

Adding the Report Viewer to a Visual Basic project

If you create a new report using the Create Report Expert in the Crystal Report Designer Component, the Report Viewer control can be automatically added to a Form in your Visual Basic project. However, there may be times when you need to add the control by hand. In addition, the Report Viewer control can be implemented in other environments, many of which may not support ActiveX designers, meaning the Create Report Expert is unavailable.

Use the following steps to add the Crystal Report Viewer ActiveX control to a Form in your Visual Basic application. This tutorial assumes the Form already exists in your project and is named Form1.

First, you must verify that a reference to the Report Viewer control exists in your project.

- 1 From the **Project** menu, select the **Components** command.
The Components dialog box appears.
- 2 On the Controls Tab of the Components dialog box, scroll through the list of ActiveX controls until you find **Crystal Report Viewer Control 9**.
If you do not see the Crystal Report Viewer control in the list, use the Browse button to locate the CRVIEWER9.DLL component in the \Program Files\Common Files\Crystal Decisions\2.0\crystalreportviewers\ActiveXViewer directory.
- 3 If the check box next to the Report Viewer control is not selected, select it on now.
- 4 Click **OK**.
The CRViewer control appears in the Visual Basic toolbox.
- 5 Click the CRViewer control on the toolbox, then draw the Report Viewer control on your form by dragging a rectangle across the form with the mouse pointer.
An instance of the control will be added to your Form.
- 6 Adjust the size and position of the Report Viewer on your form, and use the Properties window to adjust the overall appearance of the control.

Using the CRViewer object

The CRViewer object represents an instance of the Report Viewer control that has been added to your project. If you have created a report using the Crystal Report Designer Component and accepted the defaults for adding the Report Viewer to your project, the Report Viewer control in your application will be named CRViewer1. CRViewer1 can be used in your code as a CRViewer object. For instance, the following code demonstrates a simple technique for assigning a report to the Report Viewer, and displaying it:

```
CRViewer1.ReportSource = report  
CRViewer1.ViewReport
```

For more information on the properties and methods available with this object, refer to the Report Viewer object model and the CRViewer object.

The topics listed below describe several aspects of the Report Viewer object model and present examples of how to use the Report Viewer objects, methods, properties and events in your Visual Basic code.

- “Specifying a report” on page 235
- “Working with secure data in reports” on page 235
- “Handling Report Viewer events” on page 236
- “Moving through a report” on page 237
- “Printing the report” on page 238
- “Controlling the appearance of the Report Viewer” on page 238
- “Connecting to the Web Reports Server” on page 239

Specifying a report

The most important task with the Report Viewer control is to specify a report and display it at runtime. This is easily handled with the ReportSource property and the ViewReport method.

```
Private Sub Form1_Load()  
    Dim report As New CrystalReport1  
    CRViewer1.ReportSource = report  
    CRViewer1.ViewReport  
End Sub
```

In this example, assigning the report and displaying it in the Report Viewer is handled when the Form containing the Report Viewer object is loaded into the application. A reference to the report is first obtained in the form of a Report object representing a Crystal Report Designer Component that has been added to the Visual Basic project.

ReportSource is a property of the Report Viewer’s CRViewer object which corresponds directly to the Report Viewer control added to the project. In this case, that control has been named *CRViewer1*. The ReportSource property can accept a report in the form of a Report Object exposed by the Report Designer Component or the Crystal Web Reports Server.

Finally, the ViewReport method is called. This method has no parameters and has the job simply of displaying the specified report inside the Report Viewer control.

Working with secure data in reports

If your report connects to a secure data source that requires log on information, you must release the Report object from the Report Viewer before you can log off of the data source. This can be done by assigning a new Report object to the ReportSource property, or by closing the CRViewer object. Until this is done, the data source will not be released from the Report object and you cannot log off.

Handling Report Viewer events

The Report Viewer control allows you to write custom code for several events relating to user interaction with both the control window and the report displayed. For instance, if you design a drill down report using the Report Designer Component, your users are likely to want to drill down on detail data. You can provide custom handling of such an event by writing code for the **DrillOnGroup** event.

To add event procedures to the Report Viewer control for the DrillOnGroup and PrintButtonClicked events

- 1 In the Visual Basic Project window, select the Form containing the Report Viewer control.
- 2 Click the **View Code** button in the toolbar for the Project window.
A code window for the form appears.
- 3 In the drop-down list box at the upper left hand corner of the code window, select the **CRViewer1** control.
(This name will appear different if you changed the Name property of the control in the Properties window.)
- 4 In the drop-down list box at the upper right corner of the code window, select the **DrillOnGroup** event.
A procedure appears for handling the event.
- 5 Add the following code to the **DrillOnGroup** event procedure:

```
Private Sub CRViewer1_DrillOnGroup(GroupNamesList As Variant, _
    ByVal DrillType As CRVIEWERLibCtl1.CRDrillType, _
    UseDefault As Boolean)
    MsgBox "You're drilling down on the " & GroupNamesList(0) & " group!"
End Sub
```

- 6 In the drop-down list box at the upper right of the code window, select the **PrintButtonClicked** event.
A new procedure appears for this event.

- 7 Add the following code for the new event:

```
Private Sub CRViewer1_PrintButtonClicked(UseDefault As Boolean)
    MsgBox "You clicked the Print button!"
End Sub
```

The **DrillOnGroup** event is triggered when a user double-clicks a chart, a map, or a report summary field. The code added to the event procedure will display a message box with the name of the group. The **PrintButtonClicked** event is fired if the user clicks the print button on the Report Viewer window. Note that any code added to these event handlers replaces the default action of the event. A more practical use of these events would be to display custom dialogs or perform other report related calculations and procedures.

Moving through a report

Often, reports consist of several pages. The Report Viewer control provides, by default, controls that allow a user to move through the pages of the report. However, you may need to implement a system through which your own code controls when separate pages are displayed.

The CRViewer object provides several methods for moving through a report, including methods to move to specific pages:

- ShowFirstPage
- ShowLastPage
- ShowNextPage
- ShowPreviousPage
- ShowNthPage
- GetCurrentPageNumber

And methods for moving to specific groups in the report:

- ShowGroup

Moving through pages

The first set of methods designed for moving through the pages of a report are straightforward and correspond directly to controls that normally appear on the Report Viewer control window. ShowFirstPage, ShowLastPage, ShowNextPage, and ShowPreviousPage simply switch to the first, last, next, or previous page in the report, respectively. They are all used in the same manner in code:

```
CRViewer1.ShowFirstPage
CRViewer1.ShowLastPage
CRViewer1.ShowNextPage
CRViewer1.ShowPreviousPage
```

If the requested page cannot be displayed, for instance, if the last page in the report is currently displayed and ShowNextPage is called, the currently displayed page will be refreshed.

For more controlled movements through the report, ShowNthPage can display a specific page of the report:

```
CRViewer1.ShowNthPage 5
```

This method accepts a page number as its only argument. If the selected page number does not exist, for example, page 10 is selected from a 6 page report, then either the last or first page will be displayed, depending on the page number requested.

As a convenience, the GetCurrentPageNumber method has also been included. You can obtain the currently displayed page from within your code at any time using this method:

```
Dim pageNum As Long
pageNum = CRViewer1.GetCurrentPageNumber
```

Moving to a specific group

Grouping is a common feature of reports, and, since page numbers can frequently change based on current data, it may be more appropriate to navigate through a report using groups. For example, if a report is grouped by cities within states, and by states within countries, you can include code to display the group for a specific city.

Printing the report

Although the Report Viewer control is designed primarily for displaying reports on screen, users frequently want a hard-copy of the data. The PrintReport method provides a simple means of allowing access to the Windows print features. Simply call the method as below, and Windows can take over.

```
Dim Report As New CrystalReport1
CRViewer1.ReportSource = Report
CRViewer1.PrintReport
```

Controlling the appearance of the Report Viewer

By default, the Report Viewer window includes several controls for allowing users to navigate through a report, enlarge the view of a report, refresh the data in a report, and more. There may be applications that you create in which you want to limit a user's interaction, change the look of the Report Viewer window, or provide an alternate means of accessing the same functionality.

For instance, you could turn off the navigation controls in the Report Viewer, then create your own controls to navigate through the report that call the ShowFirstPage, ShowLastPage, ShowNextPage, ShowPreviousPage, and ShowNthPage methods. (See [“Moving through a report” on page 237](#).) For handling such custom features, the Report Viewer object model provides several properties for enabling and disabling different features of the Report Viewer ActiveX control:

- DisplayBackgroundEdge
- DisplayBorder
- DisplayGroupTree
- DisplayTabs
- DisplayToolbar
- EnableAnimationCtrl
- EnableCloseButton
- EnableDrillDown
- EnableExportButton
- EnableGroupTree
- EnableHelpButton
- EnableNavigationControls

- EnablePopupMenu
- EnablePrintButton
- EnableProgressControl
- EnableRefreshButton
- EnableSearchControl
- EnableSearchExpertButton
- EnableSelectExpertButton
- EnableStopButton
- EnableToolbar
- EnableZoomControl

Using these properties requires assigning a value of either True or False. True enables the specified control or feature of the Report Viewer, while False disables it.

The following code demonstrates how to disable the entire toolbar for the Report Viewer window:

```
CRViewer1.DisplayToolbar = False
```

Connecting to the Web Reports Server

The Web Reports Server provides not only a powerful means of distributing reports across the web, but also provides a report distribution mechanism that can be incorporated into multi-tier applications. By using the Crystal Report Viewer for ActiveX as a client-side report viewer, the Web Reports Server can become a report distribution engine within a larger application that runs over a network.

Connecting to the Web Reports Server requires accessing two new ActiveX components: the WebReportBroker and the WebReportSource. The following samples demonstrate how to connect to the Web Reports Server using “[Connecting from Visual Basic](#)”, and “[Connecting from VBScript](#)”, inside a web page.

Connecting from Visual Basic

The following code is an example of how to connect to the Web Reports Server from Visual Basic and assign a report to the Crystal Report Viewer for ActiveX. This assumes that you have added the ActiveX viewer control to a form named Form1, and the ActiveX viewer control is named CRViewer1.

```
Private Sub Form1_Load()
    Dim webBroker, webSource
    Set webBroker = CreateObject("WebReportBroker.WebReportBroker")
    Set webSource = CreateObject("WebReportSource.WebReportSource")
    webSource.ReportSource = webBroker
    webSource.URL = "http://<machinename>/scrreports/xtreme/hr.rpt"
    webSource.Title = "Employee Profiles"
    CRViewer1.ReportSource = webSource
    CRViewer1.ViewReport
End Sub
```

Connecting from VBScript

The following code assumes you have added the Crystal Report Viewer for ActiveX to a web page using the <OBJECT> tag and assigned it an ID of CRViewer.

```
<OBJECT ID="WebSource" Width=0 Height=0>
  CLASSID="CLSID:F2CA2115-C8D2-11D1-BEBD-00A0C95A6A5C"
  CODEBASE="viewer/ActiveXViewer/swebrs.dll#Version=1.2.0.5"
</OBJECT>
<OBJECT ID="WebBroker" Width=0 Height=0>
  CLASSID="CLSID:F2CA2119-C8D2-11D1-BEBD-00A0C95A6A5C"
  CODEBASE="viewer/ActiveXViewer/swebrs.dll#Version=1.2.0.5"
</OBJECT>
<OBJECT ID="Export" Width=0 Height=0>
  CLASSID="CLSID:BD10A9C1-07CC-11D2-BEFF-00A0C95A6A5C"
  CODEBASE="viewer/ActiveXViewer/sviewhlp.dll#Version=1.0.0.4"
</OBJECT>
<SCRIPT LANGUAGE="VBScript">
<!--
Sub Page_Initialize
  Dim webBroker
  Dim webSource
  Set webBroker = CreateObject("WebReportBroker.WebReportBroker")
  Set webSource = CreateObject("WebReportSource.WebReportSource")
  webSource.ReportSource = webBroker
  webSource.URL = Location.Protocol + "/" + Location.Host + _
    "/scrreports/xtreme/invent.rpt"
  CRViewer.ReportSource = webSource
  CRViewer.ViewReport
End Sub
-->
</SCRIPT>
```

The Crystal Report Viewer Java Bean

The Crystal Report Viewer Java Bean (or Report Viewer Bean) can be added to an application in any development environment that supports Java (version 1.1). Programmers receive the benefit of quickly adding a powerful report viewer to an application with little coding.

As a standard component, the Crystal Report Viewer Java Bean exposes several properties at design time, but also provides a complete object model with properties, methods, and events that can be programmed at runtime. The following discusses one approach to creating an application using the Crystal Report Viewer Java Bean. It describes the creation of a simple Applet which will allow a report to be viewed from your browser.

This example uses the Bean Box a component of the Bean Developer Kit (BDK) from Sun Microsystems Inc. The Bean Box is not intended to be used for serious application development, rather as a platform for testing Beans interactively at design time, and creating simple applets for run time testing. The Bean Box is available for download from Sun Microsystems.

Adding the Report Viewer Bean to the project

To add the Report Viewer Bean to the Bean Box

- 1 Locate the JAR file called **ReportViewerBean.jar** in the Viewers directory.
Program Files\Common Files\Crystal
Decisions\2.0\crystalreportviewers\JavaViewerBean
- 2 Either copy the file to the \jars subdirectory of the BDK, or from the Bean Box, select **LoadJar** from the **File** menu and specify the pathname of the file.
The Crystal Report Viewer Icon should appear in the ToolBox palette.

Creating a simple applet with the Report Viewer

To add the Report Viewer Bean to the Bean Box Composition window and create an applet

- 1 Click the Report Viewer Bean's name (Crystal Report Viewer) in the ToolBox palette.
- 2 Click the location in the Bean Box Composition window where you want the Report Viewer Bean to appear.
- 3 Resize the Report Viewer in the Composition window until you are able to see the controls and report window.

In the Bean Box Property Sheet window, you will see the list of Report Viewer Bean properties. These can be set and edited. For example to view a report click on the `reportName` property. When the dialog box appears enter the URL of a report file (for example: "`http://localhost/scrreports/craze/adcont2s.rpt`"). The report should be displayed in the Crystal Report Viewer Report window.

- 4 To create a simple applet, select **MakeApplet** from the **File** menu.
This will create an applet which when called from your browser will display the report specified in the `reportName` property. You will be prompted to specify a directory where your applet and its supporting file will be placed (or the default tmp subdirectory of the beanbox directory).

If you look at the directory containing the applet, you will notice that there are a number of supporting files and directories. Locate the HTML file (`<appletname>.html`) and click it. Your default browser displays the Report Viewer and the report.

The minimum required to actually run the application using the bean is:

- The HTML file which references the applet class file.
- The extracted ReportViewerBean.jar file and any supporting jar files.
- The applet class file.

The Crystal Report Viewer contains an extensive object model allowing you complete control over how the viewer appears and functions. This chapter provides detailed information on the properties, methods and events of the Crystal Report Viewer object model for both the ActiveX viewer, and the Java Bean viewer.

Report Viewer/ActiveX Object Model technical reference

The following diagram outlines the Report Viewer hierarchy.



CRField Object (CRVIEWERLib)

The CRField Object contains information related to the fields in a report displayed in the Report Viewer.

CRField Object Properties

Property	Description	Read/Write
FieldType	" CRFieldType (CRViewerLib)". Gets the type of field.	Read-only
IsRawData	Boolean. Gets whether or not the data in the field is raw data.	Read-only
Name	String. Gets the name of the field.	Read-only
Value	Variant. Default property gets the value in the field.	Read-only

CRFields Collection (CRVIEWERLib)

The CRFields Collection contains instances of CRFields Objects.

CRFields Collection Properties

Property	Description	Read/Write
Count	Long. Gets the total number of items in the Collection.	Read-only
Item (index As Long)	Long. Default property gets the 1-based index number of the item in the Collection.	Read-only
SelectedFieldIndex	Long. Gets the index of the selected field.	Read-only

CRVEventInfo Object (CRVIEWERLib)

The CRVEventInfo Object contains information about events relating to objects within a report.

CRVEventInfo Object Properties

Property	Description	Read/Write
CanDrillDown	Boolean. Gets whether or not the object is drillable.	Read-only
Index	Long. Gets the number identifying a control in a control array.	Read-only
ParentIndex	Long. Gets reference to the object's parent's index.	Read-only
Text	String. Gets the object's text string.	Read-only
Type	"CRObjType (CRViewerLib)". Gets the object type.	Read-only

CRVEventInfo Object Methods

GetFields Method (CRVEventInfo Object)

Use the GetFields method to get the Fields Collection.

Syntax

```
Function GetFields ()
```

CRViewer Object (CRVIEWERLib)

The CRViewer Object is the primary object representing the Report Viewer control as it appears on a Form in your Visual Basic application. The current interface is ICystalReportViewer3.

CRViewer Object Properties

Property	Description	Read/Write
ActiveViewIndex	Integer. Gets the 1-based index of the current view (tab).	Read only
DisplayBackgroundEdge	Boolean. Gets or sets whether the report is offset from the edge of its view window.	Read/Write
DisplayBorder	Boolean. Gets or sets whether the border of the viewer object is displayed.	Read/Write
DisplayGroupTree	Boolean. Gets or sets the visibility of the group tree.	Read/Write
DisplayTabs	Boolean. Gets or sets whether the viewer has tabs for navigation between views.	Read/Write
DisplayToolbar	Boolean. Gets or sets the visibility of the toolbar.	Read/Write
EnableAnimationCtrl	Boolean. Gets or sets whether or not the animation control is visible.	Read/Write

Property	Description	Read/Write
EnableCloseButton	Boolean. Gets or sets the visibility of the close button.	Read/Write
EnableDrillDown	Boolean. Gets or sets whether drill down is allowed.	Read/Write
EnableExportButton	Boolean. Gets or sets the visibility of the Export toolbar button.	Read/Write
EnableGroupTree	Boolean. Gets or sets whether or not the group tree is available.	Read/Write
EnableHelpButton	Boolean. Gets or sets whether or not the help button appears on the toolbar.	Read/Write
EnableNavigationControls	Boolean. Gets or sets whether or not the First Page button, Previous Page button, Next Page button, and Last Page button appear on the toolbar.	Read/Write
EnablePopupMenu	Boolean. Gets or sets whether the popup menu is available.	Read/Write
EnablePrintButton	Boolean. Gets or sets the visibility of the Print button.	Read/Write
EnableProgressControl	Boolean. Gets or sets the visibility of the progress control.	Read/Write
EnableRefreshButton	Boolean. Gets or sets the visibility of the Refresh button.	Read/Write
EnableSearchControl	Boolean. Gets or sets the visibility of the search control.	Read/Write
EnableSearchExpertButton	Boolean. Gets or sets the status of the Search Expert toolbar button.	Read/Write
EnableStopButton	Boolean. Gets or sets whether the viewer displays the stop button.	Read/Write
EnableToolbar	Boolean. Gets or sets the visibility of the toolbar.	Read/Write
EnableZoomControl	Boolean. Gets or sets the visibility of the zoom control.	Read/Write
IsBusy	Boolean. Gets the status of the control, busy or not busy.	Read only
ReportSource	Unknown. Gets or sets the report source.	Read/Write
TrackCursorInfo	“ CRVTrackCursorInfo Object (CRVIEWERLib) ”. Gets reference to TrackCursor information.	Read only
ViewCount	Integer. Gets the current number of views (tabs).	Read only

CRViewer Object Methods

The following methods are discussed in this section:

- “ActivateView Method (CRViewer Object)” on page 247
- “AddView Method (CRViewer Object)” on page 247
- “CloseView Method (CRViewer Object)” on page 248
- “GetCurrentPageNumber Method (CRViewer Object)” on page 248
- “GetViewName Method (CRViewer Object)” on page 248
- “GetViewPath Method (CRViewer Object)” on page 249
- “PrintReport Method (CRViewer Object)” on page 249
- “Refresh Method (CRViewer Object)” on page 250
- “SearchByFormula Method (CRViewer Object)” on page 250
- “SearchForText Method (CRViewer Object)” on page 250
- “ShowFirstPage Method (CRViewer Object)” on page 250
- “ShowGroup Method (CRViewer Object)” on page 250
- “ShowLastPage Method (CRViewer Object)” on page 251
- “ShowNextPage Method (CRViewer Object)” on page 251
- “ShowNthPage Method (CRViewer Object)” on page 251
- “ShowPreviousPage Method (CRViewer Object)” on page 251
- “ViewReport Method (CRViewer Object)” on page 251
- “Zoom Method (CRViewer Object)” on page 252

ActivateView Method (CRViewer Object)

Use the ActivateView method to activate a particular view.

Syntax

```
Sub ActivateView ( Index )
```

Parameter

Parameter	Description
Index	The 1-based index number of the view that you want to activate.

AddView Method (CRViewer Object)

Use the AddView method to add a new view tab to the Viewer.

Syntax

```
Sub AddView ( GroupPath )
```

Parameter

Parameter	Description
GroupPath	GroupPath can be a colon-delimited string (Country:State:City) or a safe array of strings. It indicates the group for which you want to add a view (tab) to the Report Viewer window.

CloseView Method (CRViewer Object)

Use the CloseView method to close the specified view.

Syntax

```
Sub CloseView ( Index )
```

Parameter

Parameter	Description
Index	The 1- based index number of the view that you want to close.

GetCurrentPageNumber Method (CRViewer Object)

Use the GetCurrentPageNumber method to retrieve the number of the page of the report that is currently being viewed.

Syntax

```
Function GetCurrentPageNumber ( ) As Long
```

Returns

Returns the current page number, if the call is successful.

GetViewName Method (CRViewer Object)

Use the GetViewName method to retrieve the current view's tab name and the current report document's name.

Syntax

```
Function GetViewName ( pTabName As String ) As String
```

Parameter

Parameter	Description
pTabName	Specifies the name of the current view (tab) As String.

Returns

- Returns the current report's document name, if the call is successful.
- Passes back the current view's tab name.

GetViewPath Method (CRViewer Object)

Use the GetViewPath method to retrieve the path to the current view. Path contains a safe array of strings. Views refer to the main Preview Tab and drill down tabs that appear in the Report Viewer as the user interacts with the report.

Syntax

```
Function GetViewPath ( Index As Integer )
```

Parameter

Parameter	Description
Index	Specifies the 1-based index number of the view (tab) displayed in the Report Viewer for which you want to retrieve the path.

Returns

Returns a safe array of strings indicating the path to the current view, if the call is successful.

Example

Use the following code as an example of how to use GetViewPath.

```
Dim vPath As Variant
Dim vString As String
Dim x As Integer
Dim y As Integer
Dim counter As Integer
vPath = CRViewer1.GetViewPath(userEnteredInteger)
x = Lbound(vPath)
y = Ubound(vPath)
For counter = x To y
    If vString <> "" Then
        vString = vString & ":"
    End If
    vString = vString & vPath(counter)
Next counter
ViewPathName.Caption = "View path is: " & vString
```

PrintReport Method (CRViewer Object)

Use the PrintReport method to initiate printing of the report in the current view.

Syntax

```
Sub PrintReport ()
```

Refresh Method (CRViewer Object)

Use the Refresh method to reload and display the report displayed in the Report Viewer from its original source.

Syntax

```
Sub Refresh ( )
```

SearchByFormula Method (CRViewer Object)

Use the SearchByFormula method to search using the specified formula. The Search GUI is displayed if parameter formula is empty.

Syntax

```
Sub SearchByFormula ( formula As String )
```

Parameter

Parameter	Description
formula	Specifies the formula that you want to use for the search As String.

SearchForText Method (CRViewer Object)

Use the SearchForText to search for the specified String.

Syntax

```
Sub SearchForText ( Text As String )
```

Parameter

Parameter	Description
Text	Specifies the text that you want to search for As String.

ShowFirstPage Method (CRViewer Object)

Use the ShowFirstPage method to display the first page of the report.

Syntax

```
Sub ShowFirstPage ( )
```

ShowGroup Method (CRViewer Object)

Use the ShowGroup method to display the indicated group in the current view. GroupPath can be a colon-delimited string (Country:State:City) or a safe array of strings.

Syntax

```
Sub ShowGroup (GroupPath)
```

Parameter

Parameter	Description
GroupPath	Specifies the path to the group that you want to display. Use a safe array of strings or a colon delimited string (for example, Canada:BC:Vancouver).

ShowLastPage Method (CRViewer Object)

Use the ShowLastPage method to display the last page of the report.

Syntax

```
Sub ShowLastPage ( )
```

ShowNextPage Method (CRViewer Object)

Use the ShowNextPage method to display the next page of the report.

Syntax

```
Sub ShowNextPage ( )
```

ShowNthPage Method (CRViewer Object)

Use the ShowNthPage method to display the specified page of the report.

Syntax

```
Sub ShowNthPage ( PageNumber As Integer )
```

Parameter

Parameter	Description
PageNumber	The page number that you want to display As Integer.

ShowPreviousPage Method (CRViewer Object)

Use the ShowPreviousPage method to display the previous page of the report.

Syntax

```
Sub ShowPreviousPage ( )
```

ViewReport Method (CRViewer Object)

Use the ViewReport method to display the report.

Syntax

```
Sub ViewReport ( )
```

Zoom Method (CRViewer Object)

Use the Zoom method to change the magnification used to display the report.

Syntax

Sub Zoom (ZoomLevel As Integer)

Parameter

Parameter	Description
ZoomLevel	The zoom level to use to view the report As Integer. Indicate a percentage, or use 1 to fit the entire width of the page in the Report Viewer window (but not the entire page) or 2 to fit the entire page in the window.

CRViewer Object Events

The following events are discussed in this section:

- “Clicked Event (CRViewer Object)” on page 253
- “CloseButtonClicked Event (CRViewer Object)” on page 253
- “DbClicked Event (CRViewer Object)” on page 253
- “DownloadFinished Event (CRViewer Object)” on page 254
- “DownloadStarted Event (CRViewer Object)” on page 254
- “DrillOnDetail Event (CRViewer Object)” on page 255
- “DrillOnGraph Event (CRViewer Object)” on page 255
- “DrillOnGroup Event (CRViewer Object)” on page 255
- “DrillOnSubreport Event (CRViewer Object)” on page 256
- “ExportButtonClicked Event (CRViewer Object)” on page 256
- “FirstPageButtonClicked Event (CRViewer Object)” on page 257
- “GoToPageNClicked Event (CRViewer Object)” on page 257
- “GroupTreeButtonClicked Event (CRViewer Object)” on page 257
- “HelpButtonClicked Event (CRViewer Object)” on page 258
- “LastPageButtonClicked Event (CRViewer Object)” on page 258
- “NextPageButtonClicked Event (CRViewer Object)” on page 258
- “OnReportSourceError Event (CRViewer Object)” on page 258
- “PrevPageButtonClicked Event (CRViewer Object)” on page 259
- “PrintButtonClicked Event (CRViewer Object)” on page 259
- “RefreshButtonClicked Event (CRViewer Object)” on page 259
- “SearchButtonClicked Event (CRViewer Object)” on page 260
- “SearchExpertButtonClicked Event (CRViewer Object)” on page 260
- “SelectionFormulaBuilt Event (CRViewer Object)” on page 260
- “SelectionFormulaButtonClicked Event (CRViewer Object)” on page 261
- “ShowGroup Event (CRViewer Object)” on page 261

- “StopButtonClicked Event (CRViewer Object)” on page 261
- “ViewChanged Event (CRViewer Object)” on page 262
- “ViewChanging Event (CRViewer Object)” on page 262
- “ZoomLevelChanged Event (CRViewer Object)” on page 263

Clicked Event (CRViewer Object)

The Clicked event occurs when an object in the viewer is clicked.

Syntax

Event Clicked (x As Long, y As Long, EventInfo, UseDefault As Boolean)

Parameters

Parameter	Description
X	Long. The X coordinate of the object clicked.
Y	Long. The Y coordinate of the object clicked.
EventInfo	A “ CRVEventInfo Object (CRVIEWERLib) ” containing information about the object clicked.
UseDefault	Boolean. Indicates whether or not the default action of the event will be performed.

CloseButtonClicked Event (CRViewer Object)

The CloseButtonClicked event occurs when the Close Current View button is clicked.

Syntax

Event CloseButtonClicked (UseDefault As Boolean)

Parameter

Parameter	Description
UseDefault	Boolean. Indicates whether or not the default action of the event will be performed.

DbClicked Event (CRViewer Object)

The DbClicked event occurs when an object is double clicked.

Syntax

Event DbClicked (x As Long, y As Long, EventInfo, UseDefault As Boolean)

Parameters

Parameter	Description
X	Long. The X coordinate of the object that was double clicked.
Y	Long. The Y coordinate of the object that was double clicked.
EventInfo	A "CRVEventInfo Object (CRVIEWERLib)" containing information about the object clicked.
UseDefault	Boolean. Indicates whether or not the default action of the event will be performed.

DownloadFinished Event (CRViewer Object)

The DownloadFinished event occurs when report data finishes loading into the report. For example, if the user displays a new page, a new set of report data is downloaded.

Syntax

```
Event DownloadFinished ( loadingType As CRLoadingType )
```

Parameter

Parameter	Description
LoadingType	"CRLoadingType (CRViewerLib)". Indicates the type of data being loaded into the Report Viewer.

DownloadStarted Event (CRViewer Object)

The DownloadStarted event occurs when report data starts being downloaded into the Report Viewer. For example, if the user displays a new page, a new set of report data is downloaded.

Syntax

```
Event DownloadStarted ( loadingType As CRLoadingType )
```

Parameter

Parameter	Description
LoadingType	"CRLoadingType (CRViewerLib)". Indicates the type of data being loaded into the Report Viewer.

DrillOnDetail Event (CRViewer Object)

The DrillOnDetail event occurs when you drill down on a field in the Detail section of the report. This event is not available in the current version of the Report Viewer, but will be enabled in a future upgrade.

Syntax

```
Event DrillOnDetail ( FieldValues, SelectedFieldIndex As Long,  
    UseDefault As Boolean )
```

Parameters

Parameter	Description
FieldValues	An array of objects containing details on the field.
SelectedFieldIndex	Long. The array index of the value in the field actually drilled on.
UseDefault	Boolean. Indicates whether or not the default action of the event will be performed.

DrillOnGraph Event (CRViewer Object)

The DrillOnGraph event occurs when you drill down on a graph.

Syntax

```
Event DrillOnGraph ( PageNumber As Long, x As Long, y As Long,  
    UseDefault As Boolean )
```

Parameters

Parameter	Description
PageNumber	Long. The page number of the report containing the graph where the event occurred.
x	Long. The X coordinate of the graph that was drilled on.
y	Long. The Y coordinate of the graph that was drilled on.
UseDefault	Boolean. Indicates whether or not the default action of the event will be performed.

DrillOnGroup Event (CRViewer Object)

The DrillOnGroup event occurs when drilling down (double-clicking) on a group field viewer window.

Syntax

```
Event DrillOnGroup ( GroupNameList, DrillType As CRDrillType,  
    UseDefault As Boolean )
```

Parameters

Parameter	Description
GroupNameList	An array containing all group names for the group drilled on.
DrillType	" CRLoadType (CRViewerLib) ". Specifies what type of object the drill event occurred on (for example, graph, group tree, map).
UseDefault	Boolean. Indicates whether or not the default action of the event will be performed.

DrillOnSubreport Event (CRViewer Object)

The DrillOnSubreport event occurs when drilling down (double-clicking) on a subreport.

Syntax

```
Event DrillOnSubreport ( GroupNameList, SubreportName As String,
    Title As String, PageNumber As Long,
    Index As Long, UseDefault As Boolean )
```

Parameters

Parameter	Description
GroupNameList	An array containing all group names for the group drilled on.
SubreportName	String. Indicates the name of the subreport that was drilled on.
Title	String. Indicates the title of the subreport that was drilled on.
PageNumber	Long. Indicates the page number that the event occurred on.
Index	Long. Indicates the index of the subreport that was drilled on.
UseDefault	Boolean. Indicates whether or not the default action of the event will be performed.

ExportButtonClicked Event (CRViewer Object)

The ExportButtonClicked event occurs when the Export button is clicked.

Syntax

```
Event ExportButtonClicked ( UseDefault As Boolean )
```

Parameter

Parameter	Description
UseDefault	Boolean. Indicates whether or not the default action of the event will be performed.

FirstPageButtonClicked Event (CRViewer Object)

The FirstPageButtonClicked event occurs when the button which navigates through the report to the first page is clicked.

Syntax

Event FirstPageButtonClicked (UseDefault As Boolean)

Parameter

Parameter	Description
UseDefault	Boolean. Indicates whether or not the default action of the event will be performed.

GoToPageNClicked Event (CRViewer Object)

The GoToPageNClicked event occurs when a user requests and goes to a specific page in the report.

Syntax

Event GoToPageNClicked (UseDefault As Boolean, PageNumber As Integer)

Parameters

Parameter	Description
UseDefault	Boolean. Indicates whether or not the default action of the event will be performed.
PageNumber	Integer. The page number of the page that is to be displayed.

GroupTreeButtonClicked Event (CRViewer Object)

The GroupTreeButtonClicked event occurs when the Group Tree button is clicked to show/hide the Group Tree in the viewer window.

Syntax

Event GroupTreeButtonClicked (Visible As Boolean)

Parameter

Parameter	Description
Visible	Boolean. Indicates whether or not the Group Tree is now visible.

HelpButtonClicked Event (CRViewer Object)

The HelpButtonClicked event occurs when the Help button is clicked.

Syntax

```
Event HelpButtonClicked ( )
```

LastPageButtonClicked Event (CRViewer Object)

The LastPageButtonClicked event occurs when the button which navigates through the report to the last page is clicked.

Syntax

```
Event LastPageButtonClicked ( UseDefault As Boolean )
```

Parameter

Parameter	Description
UseDefault	Boolean. Indicates whether or not the default action of the event will be performed.

NextPageButtonClicked Event (CRViewer Object)

The NextPageButtonClicked event occurs when the button which navigates through the report to the next page is clicked.

Syntax

```
Event NextPageButtonClicked ( UseDefault As Boolean )
```

Parameter

Parameter	Description
UseDefault	Boolean. Indicates whether or not the default action of the event will be performed.

OnReportSourceError Event (CRViewer Object)

The OnReportSourceError event occurs when the report source (assigned to the CRViewer.ReportSource property) causes an error or cannot be loaded by the Report Viewer.

Syntax

```
Event OnReportSourceError ( errorMsg As String,  
    errorCode As Long, UseDefault As Boolean )
```

Parameters

Parameter	Description
errorMsg	String. Indicates the error message.
errorCode	Long. Indicates the code or ID for the error.
UseDefault	Boolean. Indicates whether or not the default action of the event will be performed.

PrevPageButtonClicked Event (CRViewer Object)

The PrevPageButtonClicked event occurs when the button which navigates through the report to the previous page is clicked.

Syntax

Event PrevPageButtonClicked (UseDefault As Boolean)

Parameter

Parameter	Description
UseDefault	Boolean. Indicates whether or not the default action of the event will be performed.

PrintButtonClicked Event (CRViewer Object)

The PrintButtonClicked event occurs when the Print button is clicked.

Syntax

Event PrintButtonClicked (UseDefault As Boolean)

Parameter

Parameter	Description
UseDefault	Boolean. Indicates whether or not the default action of the event will be performed.

RefreshButtonClicked Event (CRViewer Object)

The RefreshButtonClicked event occurs when the Refresh button is clicked.

Syntax

Event RefreshButtonClicked (UseDefault As Boolean)

Parameter

Parameter	Description
UseDefault	Boolean. Indicates whether or not the default action of the event will be performed.

SearchButtonClicked Event (CRViewer Object)

The SearchButtonClicked event occurs when the Search button is clicked.

Syntax

```
Event SearchButtonClicked ( searchText As String, UseDefault As Boolean )
```

Parameters

Parameter	Description
searchText	String. Indicates the data being searched for.
UseDefault	Boolean. Indicates whether or not the default action of the event will be performed.

SearchExpertButtonClicked Event (CRViewer Object)

The SearchExpertButtonClicked event occurs when the Search Expert button is clicked.

Syntax

```
Event SearchExpertButtonClicked ( UseDefault As Boolean )
```

Parameter

Parameter	Description
UseDefault	Boolean. Indicates whether or not the default action of the event will be performed.

SelectionFormulaBuilt Event (CRViewer Object)

The SelectionFormulaBuilt event occurs when a new selection formula is assigned to the report. This event is not valid when the Report Viewer is used in conjunction with the Report Designer Component.

Syntax

```
Event SelectionFormulaBuilt (
    selectionFormula As String, UseDefault As Boolean )
```

Parameters

Parameter	Description
selectionFormula	String. Indicates the new selection formula assigned to the report.
UseDefault	Boolean. Indicates whether or not the default action of the event will be performed.

SelectionFormulaButtonClicked Event (CRViewer Object)

The SelectionFormulaButtonClicked event occurs when the Selection Formula button in the Report Viewer is clicked. This is not valid when the Report Viewer is assigned a report source produced by the Report Designer Component.

Syntax

```
Event SelectionFormulaButtonClicked (
    selectionFormula As String, UseDefault As Boolean )
```

Parameters

Parameter	Description
selectionFormula	String. The current selection formula that will be replaced.
UseDefault	Boolean. Indicates whether or not the default action of the event will be performed.

ShowGroup Event (CRViewer Object)

The ShowGroup event occurs when you click a group in the Group Tree.

Syntax

```
Event ShowGroup ( GroupNameList, UseDefault As Boolean )
```

Parameters

Parameter	Description
GroupNameList	An array containing all group names for the group selected.
UseDefault	Boolean. Indicates whether or not the default action of the event will be performed.

StopButtonClicked Event (CRViewer Object)

The StopButtonClicked event occurs when the user clicks the Stop button in the Report Viewer, forcing the Viewer to stop loading data from the report source.

Syntax

```
Event StopButtonClicked (
    loadingType As CRLoadingType, UseDefault As Boolean )
```

Parameters

Parameter	Description
loadingType	"CRLoadingType (CRViewerLib)". Indicates what was being loaded into the Report Viewer when the Stop button was clicked.
UseDefault	Boolean. Indicates whether or not the default action of the event will be performed.

ViewChanged Event (CRViewer Object)

The ViewChanged event occurs after the view in the Report Viewer has changed. Views refer to the main Preview Tab and drill down tabs that appear in the Report Viewer as the user interacts with the report.

Syntax

```
Event ViewChanged ( oldViewIndex As Long, newViewIndex As Long )
```

Parameters

Parameter	Description
oldViewIndex	Long. An index referring to the view the user switched from.
newViewIndex	Long. An index referring to the view the user switched to.

ViewChanging Event (CRViewer Object)

The ViewChanging event occurs when there has been a request for the view in the Report Viewer to change. Views refer to the main Preview Tab and drill down tabs that appear in the Report Viewer as the user interacts with the report.

Syntax

```
Event ViewChanging ( oldViewIndex As Long, newViewIndex As Long )
```

Parameters

Parameter	Description
oldViewIndex	Long. An index referring to the view the user is switching from.
newViewIndex	Long. An index referring to the view the user is switching to.

ZoomLevelChanged Event (CRViewer Object)

The ZoomLevelChanging event occurs when the zoom level of the Report Viewer is changed.

Syntax

Event ZoomLevelChanged (ZoomLevel As Integer)

Parameter

Parameter	Description
ZoomLevel	Integer. A value indicating the new zoom level percentage.

CRVTrackCursorInfo Object (CRVIEWERLib)

The CRVTrackCursorInfo Object contains information about the types of mouse cursors displayed while the user interacts with the report in the Report Viewer. This object corresponds to the TrackCursorInfo property in the “CRViewer Object (CRVIEWERLib)”.

CRVTrackCursorInfo Object Properties

Property	Description	Read/Write
DetailAreaCursor	“CRTrackCursor (CRViewerLib)”. Gets or sets the DetailAreaCursor type.	Read/Write
DetailAreaFieldCursor	“CRTrackCursor (CRViewerLib)”. Gets or sets the DetailAreaFieldCursor type.	Read/Write
GraphCursor	“CRTrackCursor (CRViewerLib)”. Gets or sets the GraphCursor type.	Read/Write
GroupAreaCursor	“CRTrackCursor (CRViewerLib)”. Gets or sets the GroupAreaCursor type.	Read/Write
GroupAreaFieldCursor	“CRTrackCursor (CRViewerLib)”. Gets or sets the GroupAreaFieldCursor type.	Read/Write

WebReportBroker Object (CRWEBREPORTBROKERLib)

The WebReportBroker Object is used internally by the Report Viewer to access web servers. In most applications, developers will not need to access this object directly.

WebReportSource Object (CRWEBREPORTBROKERLib)

The WebReportSource Object contains information and methods related to the display of a report by the Report Viewer.

WebReportSource Object Properties

Property	Description	Read/Write
ImageType	Reserved. Do not use in current development. Gets or sets the image type.	Read/Write
PromptOnRefresh	Boolean. Gets or sets the prompt mode. If TRUE, the user will be prompted for new information each time the Report Viewer displays the report. If FALSE, the user will not get a prompt and the information provided with the previous prompt will be used, unless the server requires new information.	Read/Write
ReportSource	Unknown. Gets or sets the report source for the WebReportBroker.	Read/Write
Title	String. Gets or sets the report title that can be displayed in viewer.	Read/Write
URL	String. Gets or sets the URL source for the report.	Read/Write

WebReportSource Object Methods

The following methods are discussed in this section:

- [“AddParameter Method \(WebReportSource Object\)” on page 264](#)
- [“AddParameterEx Method \(WebReportSource Object\)” on page 264](#)

AddParameter Method (WebReportSource Object)

Use the AddParameter method to pass additional information for setting values for prompts in code as an alternative to prompting user to provide it. For example, you can provide user information which the server might request, rather than prompting the user to enter the information at runtime.

Syntax

```
Sub AddParameter ( tag As String, value As String )
```

Parameters

Parameter	Description
tag	String. Specifies the prompt for which you want to pass a value.
value	String. Specifies the response string that you want to provide.

AddParameterEx Method (WebReportSource Object)

AddParameterEx method is not implemented for the current release. This method will allow you to pass more information than AddParameter Method, which should be used at this time.

Enumerated Types

The following enumerated types of the Report Viewer Object Model are discussed in this section:

- “CRLoadingType (CRViewerLib)” on page 265
- “CRFieldType (CRViewerLib)” on page 265
- “CRLoadingType (CRViewerLib)” on page 265
- “CRObjectType (CRViewerLib)” on page 266
- “CRTrackCursor (CRViewerLib)” on page 267

CRDrillType (CRViewerLib)

Constant	Value
crDrillOnGraph	2
crDrillOnGroup	0
crDrillOnGroupTree	1
crDrillOnMap	3
crDrillOnSubreport	4

CRFieldType (CRViewerLib)

Constant	Value
crBoolean	5
crCurrency	4
crDate	6
crDateTime	8
crInt16	1
crInt32	2
crInt8	0
crNumber	3
crString	9
crTime	7
crUnknownFieldType	255

CRLoadingType (CRViewerLib)

Constant	Value
LoadingNothing	0
LoadingPages	1

Constant	Value
LoadingQueryInfo	3
LoadingTotaller	2

CRObjType (CRViewerLib)

Constant	Value
crBitmap	103 (&H67)
crBlob	104 (&H68)
crBox	106 (&H6A)
crCrossTab	110 (&H6E)
crCrosstabChart	108 (&H6C)
crCrosstabMap	115 (&H73)
crDatabaseFieldType	1
crDetailChart	109 (&H6D)
crDetailMap	116 (&H74)
crDetailSection	202 (&HCA)
crFormulaFieldType	5
crGraphic	111 (&H6F)
crGroupChart	107 (&H6B)
crGroupFooterSection	201 (&HC9)
crGroupHeaderSection	200 (&HC8)
crGroupMap	114 (&H72)
crGroupNameFieldType	8
crLine	105 (&H69)
crOLAPChart	113 (&H71)
crOLAPCrossTabFieldType	4
crOLAPDataFieldType	3
crOLAPDimensionFieldType	2
crOLAPMap	117 (&H75)
crOLEObject	101 (&H65)
crOOPSubreport	112 (&H70)
crPageFooterSection	206 (&HCE)
crPageHeaderSection	205 (&HCD)
crPromptingVarFieldType	9
crReportFooterSection	204 (&HCC)

Constant	Value
crReportHeaderSection	203 (&HCB)
crSpecialVarFieldType	7
crSubreport	102 (&H66)
crSummaryFieldType	6
crText	100 (&H64)
crUnknownFieldDefType	0

CRTrackCursor (CRViewerLib)

Constant	Value
crAppStartingCursor	12
crArrowCursor	1
crCrossCursor	2
crDefaultCursor	0
crHelpCursor	13
crIBeamCursor	3
crMagnifyCursor	99
crNoCursor	10
crWaitCursor	11

The Report Viewer/Java Bean technical reference

The following Properties, Methods, and Events are discussed in this section:

- “The Report Viewer/Java Bean Properties” on page 267
- “The Report Viewer/Java Bean Methods” on page 270
- “The Report Viewer/Java Bean Events” on page 272

The Report Viewer/Java Bean Properties

The Report Viewer Bean properties may have one or more of the characteristics listed below:

- read: you can get the current value.
- write: you can set the value.
- bound: you can get a notification every time the value changes.
- constrained: you can veto a request to change the value.

Property	Description	Read(r), Write(w), Bound(b), Constrained(c)
busy	Boolean. True if the ReportViewer is currently processing a command initiated by user action, method call, or property change.	r, b
canCloseCurrentView	Boolean. True if the current views tab can be closed. The initial ("Preview") tab cannot be closed. Refer to method closeCurrentView.	r,b
canDrillDown	Boolean. True if drill-down views can be opened. Normally, clicking on a hidden group in the group tree (indicated by a magnifying glass icon next to the group name) or double-clicking on a chart or map or group section in the page will open a drill-down view.	r,w,b,c
currentMessage	String. The message currently displayed in the status area of the toolbar.	r,b
currentPageNumber	Integer. The number of the page currently being viewed.	r,b
currentTip	String. The "tool tip" currently displayed in the status area of the toolbar. Tool tips temporarily override any message in the status area.	r,b
currentViewName	String. The name of the view whose tab is selected.	r,b
exportingPossible	Boolean. False if exporting is not possible because the user has denied the bean permission to write to the local disk.	r
hasExportButton	Boolean. True if the Export button can be made visible in the toolbar. If exporting is not possible (refer to property <i>exportingPossible</i>), requests to set this property to True will be vetoed.	r,w,b,c
hasGroupTree	Boolean. If set to True then the GroupTree toggle button is made visible in the toolbar and the GroupTree can be displayed (refer to property <i>showGroupTree</i>).	r,w,b,c
hasPrintButton	Boolean. If True then the Print button will be visible in the toolbar. If printing is not possible (refer to property <i>printingPossible</i>) then requests to set this property to True will be vetoed.	r,w,b,c
hasRefreshButton	Boolean. If True then the Refresh button is visible in the toolbar.	r,w,b,c
hasToolBar	Boolean. If True then the toolbar is visible.	r,w,b,c
hasTextSearch Controls	Boolean. If True then the Text Search field and the Find Next button are visible in the toolbar.	r,w,b,c

Property	Description	Read(r), Write(w), Bound(b), Constrained(c)
language	String. Contains the 2 letter international Standard code for the language to be used for the user interface. Languages currently supported are: English -- en French -- fr German -- de Japanese -- ja Italian -- it Spanish -- es Portuguese -- pt	r,w,b,c
lastPageNumber	Integer. Indicates to the highest numbered page currently available in the report. This may or may not be the final page. (refer to property <i>lastPageNumberKnown</i>).	r,b
lastPageNumberKnown	Boolean. True if the number of the final page in the report is known. If False then there are more pages in the report than the lastPageNumber property indicates.	r,b
printingPossible	Boolean. True if printing is possible. If False then either the Java implementation doesn't support it or the user has denied the bean permission to print.	r
reportName	String. The URL of the report to be viewed. For example: http://server_name/report_dir/report.rpt Setting this property causes the ReportViewer to request page 1 of the report from the server.	r,w,b,c
searchText	String. Contains the text most recently searched for, or the text being typed by the user into the Text Search field in the toolbar.	r,b
selectionFormula	String. The current selection formula to be used for subsequent commands. The formula is expressed in the Crystal Reports formula language. Setting this property closes all views except the initial one (the "Preview" view), discards all information cached for the report, and re-requests the current page of the report.	r,w,b,c
showGroupTree	Boolean. If True and the <i>hasGroupTree</i> property is True then the GroupTree will be visible.	r,w,b,c

The Report Viewer/Java Bean Methods

Each of the methods in the Report Viewer Bean starts what may be a lengthy operation, and they return to the caller before that operation is complete. If it is important to know when the command begun by one of these methods is finished, the calling code should watch for the associated events or property change notifications.

Generally there will be a time delay between the method call returning and the associated event being fired or the property changing. In fact, there may be a delay in beginning the command if the report viewer is busy processing a previous command. Commands are begun strictly one at a time in the order they are generated although, once begun, they may be processed in parallel in different threads.

The events and property change notifications will be given to the calling code on a different thread from the one that made the method call.

closeCurrentView

If the *canCloseCurrentView* property is True, closes the current view. Equivalent to the Close button in the toolbar. A *viewClosed* and a *viewActivated* event are fired. The *currentViewName* property is changed.

Syntax

```
void closeCurrentView ( )
```

exportView

If the *exportingPossible* property is True, requests the report from the server in the indicated format and writes it to the local disk. Similar to the Export button in the toolbar.

Syntax

```
void exportView ( int exportFormat, File destinationFile )
```

Parameters

exportFormat	Specifies the format in which the requested report should appear.	
	Constant	Value
	toHTML	0
	toCrystalReport	1
	toMSWord	2
	toMSExcel	3
destinationFile	Complete pathname of the destination file.	

printView

If the *printingPossible* property is True, prints all pages in the current view, requesting them from the server if necessary. Equivalent to the Print button in the toolbar.

Syntax

```
void printView ()
```

refreshReport

Closes all views except the initial one (the “Preview” view), discards all information cached for the report, and re-requests the current page of the report.

Syntax

```
void refreshReport ()
```

searchForText

Displays the next occurrence of the indicated text in the report output. Equivalent to the text search field in the toolbar. Currently, the second and third parameters are ignored; the search is always forward and case-insensitive. The *searchText* property is changed.

Syntax

```
void searchForText ( String searchString, boolean forwardSearch,  
                    boolean caseSensitive )
```

Parameters

searchString	The string for which you want to search.
forwardSearch	This parameter is ignored.
caseSensitive	This parameter is ignored.

showLastPage

Shows the last page of the report, requesting it from the server if necessary. Equivalent to the Last Page button in the toolbar. The *lastPageNumber* and *lastPageNumberKnown* properties may be changed.

Syntax

```
void showLastPage ()
```

showPage

Displays the indicated page, requesting it from the server if necessary. Equivalent to the page number field in the toolbar. The *currentPageNumber* property is changed.

Syntax

```
void showPage ( int pageNumber )
```

Parameter

pageNumber	Number of the page to be displayed.
------------	-------------------------------------

stopAllCommands

Cancels all unfinished commands. Equivalent to the Stop button in the toolbar.

Syntax

```
void stopAllCommands ()
```

The Report Viewer/Java Bean Events

There are 2 event classes defined by the Report Viewer Bean and discussed in this section: class *ViewChangeEvent* and class *ServerRequestEvent*.

ServerRequestEvent

Description	Properties	ListenerInterface Methods
Indicates a request has been sent to the server owning the current report.	string ServerURL string Parameter	requestStarted(event e) requestEnde(event e)

ViewChangeEvent

Description	Properties	ListenerInterface Methods
Indicates that a view has changed.	string viewName	viewOpened(event e) viewActivated(event e) viewClosed(event e)

Programming the Embeddable Crystal Reports Designer Control

6

This chapter demonstrates how to integrate the Embeddable Crystal Reports Designer Control (Embeddable Designer) into your application. Included are tutorials on creating sample applications in both Microsoft Visual Basic and Microsoft Visual C++. A brief section on creating reports in the designer at runtime is also included.

Overview

The Embeddable Crystal Reports Designer Control (Embeddable Designer) is a new addition to the Report Designer Component. Easily integrated into your application, the Embeddable Designer provides your users with an interactive environment to design or edit Crystal Reports.

The following examples demonstrate how to create a sample application using the Embeddable Designer in both Microsoft Visual Basic and Microsoft Visual C++.

Two tutorials are provided:

- [“Creating a Microsoft Visual Basic sample application with the Embeddable Designer” on page 274](#)
- [“Creating a Microsoft Visual C++ sample application with the Embeddable Designer” on page 280](#)

Creating a Microsoft Visual Basic sample application with the Embeddable Designer

This tutorial assumes that you have prior knowledge of Microsoft Visual Basic, Crystal Reports, and the Report Designer Component. The following procedure and sample code detail how to create an application that demonstrates the Embeddable Designer. The application will consist of a single form with a tab control and buttons to create a new report or open an existing report. The tab control has two property pages. The first page (Designer tab) contains the Embeddable Designer, and the second page (Viewer tab) contains the Crystal Reports Viewer Control. See [“Programming the Crystal Report Viewers” on page 231](#). With the application running, you will be able to open a new report or an existing report in the Embeddable Designer. After designing or editing a report, view it in the Crystal Report Viewer. When you go back to the Embeddable Designer to make changes, you’ll see those changes updated automatically in the Crystal Report Viewer.

Note: The sample application was created in Microsoft Visual Basic version 6.0 with Service Pack 4.

This tutorial contains the following steps:

- [“Step 1: Creating the user interface” on page 275](#)
- [“Step 2: Writing the code” on page 276](#)
- [“Step 3: Running the Embeddable Designer application” on page 279](#)

Step 1: Creating the user interface

In this step, you will add the references, components, forms, and controls for the application.

- 1 Create a new **Standard EXE** project.
- 2 Add the following components to the project:
 - Crystal Reports Viewer Control.
 - Embeddable Crystal Reports 9 Designer Control.
 - Microsoft Common Dialog 6.0.
 - Microsoft Tabbed Dialog Control 6.0 (Sp4).
- 3 Add the following reference to the project:
 - Crystal Reports 9 ActiveX Designer Design and Runtime Library.

Note: See *"Programming the Crystal Report Viewers"* on page 231 for more information.
- 4 Add the following controls from the Toolbox to the form:
 - Three CommandButton controls.
 - Common Dialog (Microsoft Common Dialog Control 6.0).
 - SStab1 (Microsoft Tabbed Dialog Control 6.0 (SP4)).
 - CRDesignerCtrl (Embeddable Crystal Reports 9 Designer Control).
 - CRViewer (Crystal Report Viewer Control).
- 5 Set the properties for the form and the controls:
 - **Form1** (Main form for the application):
 - Name = frmMain
 - Caption = Embeddable Crystal Reports 9 Designer Control Sample
 - Height = 8265
 - Width = 10815
 - **SStab** (Tabbed control containing the Embeddable Designer and Report Viewer controls):

Note: Click the tabs on SStab1 to access the separate Caption properties.

 - Name = SStab1
 - Tabs = 2
 - Tab1 Caption = Design
 - Tab2 Caption = Preview
 - Enabled = False
 - Height = 7600
 - Left = 0
 - Tab Height = 300
 - Top = 130
 - Width = 10825

- **CRDesignerCtrl1** (Designs and edits reports):

Note: Place the control on the tab labeled "Designer."

- Name = CRDesignerCtrl1
- Height = 6540
- Left = 0
- Top = 0
- Width = 10325
- **CRViewer** (Views reports):
 - Name = CRViewer1
 - Height = 10340
 - Left = 120
 - Top = 360
 - Height = 6600
- **CommonDialog** (Creates an Open dialog to select reports):
 - Name = CommonDialog1
 - Filter = Crystal Reports | *.rpt
- **Button1** (Creates a new report):
 - Name = cmdNewReport
 - Caption = &New Report
 - Left = 120
 - Top = 7850
- **Button2** (Opens an existing report):
 - Name = cmdOpenReport
 - Caption = &OpenReport
 - Left = 1950
 - Top = 7850
- **Button3** (Closes the form and exits the application):
 - Name = cmdExit
 - Caption = E&xit
 - Left = 9030
 - Top = 7850

Step 2: Writing the code

In this step you will add the code to:

- Create a new report.
- Create an Open dialog box to open an existing report.
- Set the report object to CRDesignerCtrl1 and view the report in design mode.
- Set the report object to CRViewer1, set the zoom level, and view the report.
- Refresh the report when switching from the Designer tab to the Viewer tab on the Microsoft Tabbed Dialog Control.

- 1 Type or insert the sample code below into the code module of frmMain.
- 2 Once you have added the code, on the **Run** menu click **Start** to run the application.

Note: Error handling is not included in the code samples.

```
Option Explicit

Dim m_Application As New CRAXDDRT.Application
Dim m_Report As CRAXDDRT.Report

' *****
'DisplayReport is a procedure that
' - Enables the Tab control the first time a report is created
'   or opened.
' - Sets the report object to the Embeddable Designer(CRDesigner1).
' - Disables the Help menu in the Embeddable Designer.
' - Sets the report object to the Crystal Report Viewer Control
'   (CRViewer1).
' - Sets the Crystal Reports Viewer to view the report.

Public Sub DisplayReport()
' Enable the tab control if disabled.
If SSTab1.Enabled = False Then SSTab1.Enabled = True

' Set the Report Object.
CRDesignerCtrl1.ReportObject = m_Report

' Note-----
' Set all other properties for CRDesignerCtrl1 after setting the
' ReportObject property
' -----

' Disable the Help menu.
CRDesignerCtrl1.EnableHelp = False

' Set the report source.
CRViewer1.ReportSource = m_Report

' Set the viewer to view the report.
CRViewer1.ViewReport
' Set the zoom level to fit the page
' to the width of the viewer window.
CRViewer1.Zoom 1
End Sub

' *****
Private Sub Form_Load()
' Set the tab control to display the Designer tab
'when the form is loaded.
SSTab1.Tab = 0
End Sub
```

```
' *****
Private Sub SStab1_Click(PreviousTab As Integer)
' Refresh the report when clicking Preview,
' without refreshing the data from the server.
If PreviousTab = 0 Then CRViewer1.RefreshEx False

End Sub

' *****
' Create a new report and display it in the Embeddable Designer

Private Sub cmdNew_Click()
' Set the report object to nothing.
Set m_Report = Nothing

' Create a new report.
Set m_Report = m_Application.NewReport

' Call DisplayReport to set the report to the Embeddable Designer
' and the Crystal Report Viewer and then display the report in the
' Embeddable Designer.
Call DisplayReport
End Sub

' *****
' Use the Microsoft Common Dialog control to open a report.

Private Sub cmdOpen_Click()

CommonDialog1.CancelError = True

On Error GoTo errHandler

' Display the open dialog box.
CommonDialog1.ShowOpen

' Set the report object to nothing.
Set m_Report = Nothing

' Open the selected report.
Set m_Report = m_Application.OpenReport(CommonDialog1.FileName, 1)

' Call DisplayReport to set the report to the Embeddable Designer
' and the Crystal Report Viewer
Call DisplayReport

Exit Sub

errHandler:
' User cancelled dialog
End Sub
```



```

' *****
Private Sub cmdAbout_Click()
frmAbout.Show vbModal
End Sub

' *****
Private Sub cmdExit_Click()
Unload Me
End Sub

```

Step 3: Running the Embeddable Designer application

In this step you will:

- Create and design a new report in the Embeddable Designer.
 - View the report and any changes made to the report.
 - Open and edit an existing report in the Embeddable Designer.
 - View the report and any changes made to the report.
- 1 With the application running, click **New Report**.
An empty report will appear in the Embeddable Designer.
Note: The interface for the Embeddable Designer is the same one used for ActiveX Designer reports created in the Microsoft Visual Basic IDE with the Crystal Reports Report Designer Component.
 - 2 Design a new report to view.
If you are not familiar with the designer environment, see [“Designing reports in the Embeddable Designer” on page 288](#) for a step-by-step procedure on creating a simple report off the Xtreme Sample Database ODBC data source.
 - 3 Click **Preview** to view the report in the Crystal Report Viewer.
 - 4 Click **Design** and make some changes to the report. Then click **Preview** to see the changes in the Crystal Report Viewer.
 - 5 Click **Open**.
 - 6 In the Open dialog box, select one of the sample reports and click **Open** to view the report in the Embeddable Designer.
 - 7 Click **Preview** to view the report in the Crystal Report Viewer.
 - 8 Click **Design** and make some changes to the report. Then click **Preview** to see the changes in the Crystal Report Viewer.

Creating a Microsoft Visual C++ sample application with the Embeddable Designer

This tutorial assumes that you have prior knowledge of Microsoft Visual C++, Crystal Reports, and the Report Designer Component. The following procedure and sample code create an application that demonstrates the Embeddable Designer. The application will consist of a main dialogs for the Embeddable Designer, and for the Crystal Reports Viewer Control. For details, see [“Programming the Crystal Report Viewers” on page 231](#).

With the application running, you will be able to open a new or existing report in the Embeddable Designer. After designing or editing a report, you can preview it in the Crystal Report Viewer by clicking the Preview button. You can then return to the Embeddable Designer by clicking the Design button. When you go back to to edit the report, you can see those changes updated automatically by clicking the Preview button again.

This tutorial contains the following steps:

- [“Step 1: Creating the User Interface” on page 280](#)
- [“Step 2: Adding member variables and member functions” on page 282](#)
- [“Step 3: Writing the code” on page 284](#)
- [“Step 4: Running the Embeddable Designer application” on page 287](#)

Step 1: Creating the User Interface

In this step, you will create a Dialog-based MFC.EXE application and add the components, and controls for the application.

- 1 Using the AppWizard, create a starter Dialog-based MFC application. The main dialog will display reports in Embeddable Designer and the Crystal Report Viewer:
 - Project Name: Embeddable_Designer
 - Application Class Name: CEmbeddable_DesignerApp
 - Dialog Class Name: CEmbeddable_DesignerDlg
- 2 Add the Embeddable Designer Control and the Crystal Report Viewer Control through the Components and Controls Gallery dialog box. The controls are in the Registered ActiveX Controls folder:
 - Embeddable Crystal Reports 9 Designer Control
 - Crystal Report Viewer Control
- 3 Add the following controls from the Toolbox to **IDD_EMBEDDABLE_DESIGNER_DIALOG** (Embeddable Designer dialog):
 - Six buttons
 - Embeddable Crystal Reports 9 Designer
 - Crystal Report Viewer

- 4 Set the properties for the Embeddable Designer dialog and its controls:
 - **IDD_EMBEDDABLE_DESIGNER_DIALOG** (Create, open, save, design and preview reports):
 - Caption: Embeddable Designer
 - Height: 340
 - Width: 540
 - **Embeddable Crystal Reports 9 Designer Control** (Designs and edits reports):
 - Height: 300
 - Left: 7
 - Top: 7
 - Width: 525
 - **Crystal Report Viewer:**
 - Height: 300
 - Left: 7
 - Top: 7
 - Width: 525

Note: Either the Embeddable Designer or the Crystal Report Viewer will be hidden depending on whether the user is designing or previewing a report.

- **All Buttons** (Common Properties):
 - Align all Buttons to the bottom of the form
 - Height: 14
 - Width: 50
- **Button1** (Creates a new report):
 - ID: IDC_NEW_REPORT
 - Caption: &New Report
 - Left: 7
- **Button2** (Opens an existing report):
 - ID: IDC_OPEN_REPORT
 - Caption: &Open Report
 - Left: 66
- **Button3** (Saves a report):
 - ID: IDC_SAVE_REPORT
 - Caption: &Save Report
 - Left: 125
- **Button4** (Shows the Crystal Report Viewer and hides the Embeddable Designer.):
 - ID: IDC_PREVIEW
 - Caption: &Preview
 - Left: 184

- **Button5** (Shows the Embeddable Designer and hides the Crystal Report Viewer):
 - ID: IDC_SHOW_DESIGNER
 - Caption: &Design
 - Left: 184
- **Button6** (Exits the application):
 - ID: IDCANCEL
 - Caption: E&xit
 - Left: 483

Step 2: Adding member variables and member functions

In this step, you will add the member variables for the controls on the Embeddable Designer dialog and then add member functions, and member variables for the CEmbeddable_DesignerDlg class.

This step is broken into 2 sections:

- “Adding member variables for the Embeddable Designer dialog controls” on page 282
- “Adding member variables and functions to the CEmbeddable_Designer class” on page 283

Adding member variables for the Embeddable Designer dialog controls

Use the Class Wizard to create the member variables for the controls.

- 1 Create member variables for the controls on the Embeddable Designer dialog:
 - **Embeddable Designer Control:**
 - Control ID: IDC_EMBEDDABLECRYSTALREPORTSDESIGNERCTRL
 - Type: CCRDesignerCtrl
 - Member: m_Designer
 - **Crystal Report Viewer Control:**
 - Control ID: IDC_CRVIEWER1
 - Type: CCrystalReportViewer4
 - Member: m_Viewer
 - **New Report Button:**
 - Control ID: IDC_NEW_REPORT
 - Type: CButton
 - Member: m_NewReport
 - **Open Report Button:**
 - Control ID: IDC_OPEN_REPORT
 - Type: CButton
 - Member: m_OpenReport

- **Save Report Button:**
 - Control ID: IDC_SAVE_REPORT
 - Type: CButton
 - Member: m_SaveReport
- **Preview Button:**
 - Control ID: IDC_PREVIEW
 - Type: CButton
 - Member: m_Preview
- **Design Button:**
 - Control ID: IDC_SHOW_DESIGNER
 - Type: CButton
 - Member: m_ShowDesigner

Adding member variables and functions to the CEmbeddable_Designer class

Add the following member variables, and member functions to the CEmbeddable_Designer class. The code for the functions will be added in “[Step 3: Writing the code](#)” on page 284.

- 1 Create the following member variables:
 - Application pointer for the Craxddrt Application Object:
 - Variable Type: IApplicationPtr
 - Variable Name: m_Application
 - Report pointer for the Craxddrt Report Object:
 - Variable Type: IReportPtr
 - Variable Name: m_Report
- 2 Create the following member function:
 - Function to create or open a report and initialize the controls on the Embeddable Designer form.
 - Function Type: void
 - Function Declaration: InitReport(BOOL bNew)
- 3 Add a member function for the BN_CLICKED message of each of the following buttons:
 - **New Report:** (Click to create a new report)
 - Member function name: OnNewReport
 - **Open Report:** (Click to open an existing report)
 - Member function name: OnOpenReport
 - **Save Report:** (Click to save the current report)
 - Member function name: OnSaveReport
 - **Preview:** (Click to Preview the current report)
 - Member function name: OnPreview

- **Design:** (Click to edit the current report)
 - Member function name: OnShowDesigner

Step 3: Writing the code

In this step, you will write the code for member functions added in “[Step 2: Adding member variables and member functions](#)” on page 282; you will also add directives and constants to the source and header files.

This step is broken into 2 sections:

- “[Adding directives and constants to the source and header files](#)” on page 284
- “[Adding the code for the CEmbeddable_Designer class](#)” on page 284

Adding directives and constants to the source and header files

Add the following directives and constants to the Source and Header files.

- 1 Add a constant to Embeddable_DesignerDlg.cpp.

```
// Filter for the Open dialog box and the Save As dialog box
static const char BASED_CODE szFilter[] = "Crystal Reports|*.rpt|";
```

Adding the code for the CEmbeddable_Designer class

Type or insert the code for the member functions created in “[Adding member variables and functions to the CEmbeddable_Designer class](#)” on page 283. The CEmbeddable_DesignerDlg class handles the basic functionality of the Embeddable Designer sample application. The class will handle all of the report functionality including creating a new report or opening an existing report, and displaying it in the Embeddable Designer or the Crystal Report Viewer. Once a report is designed or edited it can be saved, previewed, or refreshed.

Note: Error handling is not included in the code samples.

- 1 Add the code to the OnInitDialog function.

```
// Hide the Crystal Reports Viewer control and show
// the Embeddable Designer control.
m_Viewer.ShowWindow(SW_HIDE);
m_Designer.ShowWindow(SW_SHOW);
```

- 2 Add the code for the InitReport function.

```
// In this function a report is created or opened. The report is set to
// the Embeddable Designer and Crystal Report Viewer, with the Embeddable
// Designer visible and the Crystal Report Viewer hidden. Then the
// command buttons are initialized so that Save Report, and Preview are
// enabled, and Design is disabled and hidden.
void CEmbeddable_DesignerDlg::InitReport(BOOL bNew)
{
    // Create a new report for designing and previewing.
```

```

if(bNew)
{
    m_Application.CreateInstance("CrystalDesignRuntime.Application");
    m_Report = m_Application->NewReport();
}

// Show an Open dialog box, and then browse and open a report
// for editing and previewing.
else
{
    // Initialize the Open dialog box to only display the
    // Crystal Report(.rpt) file type.
    CFileDialog FileDlg(TRUE,"*.rpt",NULL, OFN_EXPLORER, szFilter);

    // Show the Open dialog box
    int iReturn = FileDlg.DoModal();

    if(iReturn == IDOK)
    {
        // Get the name and path of the report file selected from the
        // Open dialog box.
        _bstr_t FileName(FileDlg.GetPathName().AllocSysString());

        // Create the Application Object.

        m_Application.CreateInstance("CrystalDesignRuntime.Application");

        // Open the report selected from the Open dialog box.
        m_Report = m_Application->OpenReport(FileName);
    }
    else if (iReturn == IDCANCEL)
    {
        // Terminate the execution of the function if the
        // user cancels.
        return;
    }
}

// Set the report object to the Embeddable Designer.
m_Designer.SetReportObject(m_Report);

// Set the report source for the Crystal Reports Viewer.
m_Viewer.SetReportSource(m_Report);

// Set the width of the report page to the width of the viewing area.
m_Viewer.Zoom(1);

// View the report in the Crystal Reports Viewer control.
m_Viewer.ViewReport();

// Check if the Save Report button is enabled.
// If the button is disabled then enable the Save Report,
// and Preview buttons. The Save Report button will always

```

```
// be enabled once the first report is opened or created.
BOOL bEnabled;

if( bEnabled == FALSE)
{
    m_SaveReport.EnableWindow(TRUE);
    m_Preview.EnableWindow(TRUE);
}

// Check if the Embeddable Designer is hidden.
// If it is hidden then hide the Design button, show the
// Preview button, hide the Crystal Reports Viewer, and
// show the Embeddable Designer.
if(!m_Designer.IsWindowVisible())
{
    m_ShowDesigner.ShowWindow(SW_HIDE);
    m_Preview.ShowWindow(SW_SHOW);
    m_Viewer.ShowWindow(FALSE);
    m_Designer.ShowWindow(TRUE);
}
}
```

3 Add the code for the OnNewReport function.

```
// In this function call the InitReport function and pass a
// value of True to create a new report. Passing a value of
// False opens an existing report.
void CEmbeddable_DesignerDlg::OnNewReport()
{
    InitReport(TRUE);
}
}
```

4 Add the code for the OnOpenReport function.

```
// In this function call the InitReport function and pass
// a value of False to open an existing report. Passing a
// value of True creates a new report.
void CEmbeddable_DesignerDlg::OnOpenReport()
{
    InitReport(FALSE);
}
}
```

5 Add the code for the OnSaveReport function.

```
// In this function open a Save As dialog box and save the current
// report to the selected path and name.
void CEmbeddable_DesignerDlg::OnSaveReport()
{
    // Initialize the Save As dialog box to only display the Crystal
    // Report (.rpt) file type.
    CFileDialog FileDlg(FALSE, "*.rpt", NULL, OFN_EXPLORER, szFilter);
```



```

// Show the Save As dialog box.
int iReturn = FileDlg.DoModal();

if(iReturn == IDOK)
{
    // Get the path and name selected in the Save As dialog box.
    _bstr_t FileName(FileDlg.GetPathName().AllocSysString());

    // Save the report to the path and name specified in the
    // Save As dialog box.
    m_Designer.SaveReport(FileName);
}
else if(iReturn == IDCANCEL)
{
    // Terminate the execution of the function if the
    // user cancels.
    return;
}
}

```

6 Add the code for the OnPreview function

```

// In this function, refresh the report without refreshing the data
// source, hide the Preview button and the Embeddable Designer, and
// then show the Design button and the Crystal Report Viewer.
void CEmbeddable_DesignerDlg::OnPreview()
{
    m_Viewer.RefreshEx(FALSE);
    m_Preview.ShowWindow(SW_HIDE);
    m_Designer.ShowWindow(SW_HIDE);
    m_ShowDesigner.ShowWindow(SW_SHOW);
    m_Viewer.ShowWindow(SW_SHOW);
}

```

7 Add the code for the OnShowDesigner function.

```

// In this function, hide the Design button and the Crystal Report
// Viewer, and show the Preview button and the Embeddable Designer.
void CEmbeddable_DesignerDlg::OnShowDesigner()
{
    m_ShowDesigner.ShowWindow(SW_HIDE);
    m_Viewer.ShowWindow(SW_HIDE);
    m_Preview.ShowWindow(SW_SHOW);
    m_Designer.ShowWindow(SW_SHOW);
}

```

Step 4: Running the Embeddable Designer application

In this step you will:

- Create and design a new report in the Embeddable Designer.
- View the report and any changes made to the report.
- Open and edit an existing report in the Embeddable Designer.

- View the report and any changes made to the report.
 - Save a report.
- 1 With the application running, click **New Report**.
An empty report will appear in the Embeddable Designer.
Note: The interface for the Embeddable Designer is the same one used for ActiveX Designer reports created in the Microsoft Visual Basic IDE with the Report Designer Component.
 - 2 Design a new report to view.
If you are not familiar with the designer environment, refer to “[Designing reports in the Embeddable Designer](#)” on page 288 for a step-by-step procedure on creating a simple report off the Xtreme Sample Database ODBC data source.
 - 3 Click **Preview** to view the report in the Crystal Report Viewer.
 - 4 Click **Design** and make some changes to the report in the Embeddable Designer. Then click **Preview** to see the changes in the Crystal Report Viewer.
 - 5 Click **Open**. In the Open dialog box, select one of the sample reports and click **Open** to view the report in the Embeddable Designer dialog.
 - 6 Click **Preview** to view the report in the Crystal Report Viewer.
 - 7 Click **Design** and make some changes to the report in the Embeddable Designer. Then click **Preview** to see the changes in the Crystal Report Viewer.
 - 8 Click **Save Report**.
 - 9 In the Save As dialog box, browse to the desired directory, type the name you want to save the report as, and then click **Save**.

Note: When you are designing a report the Preview button will be visible, and when you are previewing a report the Design button will be visible.

Designing reports in the Embeddable Designer

Designing reports in the Embeddable Designer is an intuitive process. In this procedure, you will create a simple report off the Xtreme Sample Database ODBC data source, which installs with Crystal Reports. The report will include a text object, database fields, and a group.

The procedure contains the following steps:

- “[Step 1: Adding a data source to the report](#)” on page 289
- “[Step 2: Adding fields and grouping to the report](#)” on page 289

Step 1: Adding a data source to the report

In this step, you will use the Data Explorer to add the Customer table from the ODBC data source Xtreme Sample Database.

- 1 On the **Main Report** tab, right-click **Database Fields**, and then click **Add Database to Report**.
- 2 In the Data Explorer dialog box, expand **ODBC** and expand **Xtreme Sample Database**; then select **Customer**, click **Add**, and click **Close**.
- 3 In the Visual Linking Expert dialog box, click **OK**.
The data source is now added to the report.

Step 2: Adding fields and grouping to the report

In this step you will add a text object to the Report Header section, the Customer Name and Last Year's Sales Fields to the Details section, and a group based on the Region field.

- 1 On the **Main Report** tab, right-click in the white space to access the shortcut menu.
- 2 On the shortcut menu, point to **Insert**, and then click **Text Object**.
An object frame appears with the Arrow pointer.
- 3 Drag the object frame to the **Report Header** section and click to release the text object.
- 4 Double-click the text object to edit it, and then type: Xtreme Sales Report by Region.
- 5 Expand **Database Fields**, and then expand **Customer** to view the database fields.
- 6 Drag the **Customer Name** and **Last Year's Sales** fields into the **Details** section.
A column heading for each field is automatically placed in the Page Header section.
- 7 Click **Insert Group** on the toolbar.
- 8 In the Insert Group dialog box, select **Region**, and then click **OK**.

You are now ready to preview, export, print, or save the report. See *Crystal Reports Online Help (crw.chm)* for more information on designing reports in the Crystal Reports environment.

Note: You can find the typical Crystal Report Designer commands (menu items and toolbar buttons) on the shortcut menu of the Embeddable Designer. The Embeddable Designer does not support OLAP grids or maps.

Embeddable Crystal Reports Designer Control Object Model

7

The Embeddable Crystal Reports Designer Control (Embeddable Designer) is a new addition to the Report Designer Component; it enables you to provide a Crystal Reports designer in your application. In this chapter, you will find a general overview of the control as well as detailed information on its properties and methods.

Overview of the Embeddable Designer

Your users can now design or edit reports at runtime with the Embeddable Designer. This easy-to-use ActiveX control provides the full power of Crystal Reports to your application. The control is placed on a form, or in a container within the form, and displays a new or existing report in design mode. This presents the user with an interactive environment in which to design or edit the report. The Embeddable Designer allows users to:

- Add data sources.
- Add database fields.
- Create and add formulas.
- Create and add parameters.
- Group data.
- Summarize data.
- Add charts.
- Add subreports.
- Add special fields.
- Format fields and sections.

Application development with the Embeddable Designer

The Embeddable Designer is an ActiveX control that can be added to any development environment that supports ActiveX. Using Microsoft Visual Basic, Microsoft Visual C++, Delphi, and Borland C++, you can now add a Crystal Reports designer to your application with minimal code.

As a standard component, the Embeddable Designer not only exposes several properties at design time, but also provides a complete set of properties, methods, and events that can be programmed at runtime. An application using the Embeddable Designer will consist of three components. These three components allow the user to create, design, preview, print, export, and save their reports.

- The *Embeddable Designer*, also known as the *Embeddable Crystal Reports 9 Designer Control* (Crdesignerctrl.dll), is a front-end user interface for designing or editing new or existing reports. See [Programming the Embeddable Crystal Reports Designer Control](#) for tutorials on creating applications with the Embeddable Designer.
- The *Automation Server*, also known as the *Crystal Reports 9 ActiveX Designer Design and Runtime Library* (Craxddrt.dll), is an extensive object model with hundreds of properties and methods that you can use to program a report. See [“Report Designer Component Object Model” on page 13](#) for more information.
Note: The Craxddrt.dll must only be used in a client-side application. See [“Unification of the RDC object model” on page 15](#) for more information.

- The *Report Viewer*, also known as the *Crystal Reports Viewer Control* (Crvviewer.dll), is a front-end user interface for viewing reports. See [“Programming the Crystal Report Viewers” on page 231](#) for more information.

Distributing the Embeddable Designer

Distribution of an application consists of three main DLLs: Crdesignerctrl.dll, Craxddrt.dll, and Crviewer.dll. In addition to these main DLLs, you will need to distribute the standard runtime files, including database DLLs and export DLLs. The additional DLLs you distribute will vary depending on your application. For more information on standard runtime files, refer to the *Crystal Reports Developer Runtime Help* (Runtime.hlp).

Note: The Embeddable Designer is not royalty free; it requires licensing. For more information on licensing, refer to the *License Manager Help* (License.hlp).

Providing online help for the Embeddable Designer

Although online help for designing reports at runtime in the Embeddable Designer is not provided, you can provide your own online help to your users. The Embeddable Designer has access to online help through the Help button on the main toolbar and through the Help command in the shortcut menu. You can enable or disable the Help button and hide the Help menu through the EnableHelp property (Microsoft Visual Basic) or the SetEnableHelp function (Microsoft Visual C++). To set your help file to the Embeddable Designer, create the following registry subkey:

Key Name

HKEY_LOCAL_MACHINE\SOFTWARE\Crystal Decisions\9.0\Report Designer Component

Value Name

CRDesignerCtlHelpPath

Value Data

Provide the full path and name of your help file here.

The Embeddable Designer uses the WinHelp function to start Microsoft Windows Help. The following table lists the various help options available and the UCommand used in the WinHelp function based on the help option selected. Keep this in mind when creating your online help.

Note: The online help must be a WinHelp file (.hlp).

Option Location	UCommand
Menu: Help > Crystal Reports Help	HELP_INDEX
Menu: Help > Search	HELP_FINDER
Menu: Help > Using Help	HELP_HELPONHELP
Toolbar: Help	HELP_INDEX

For a complete description of the WinHelp function see “WinHelp Shell and Common Controls: Platform SDK” in the Microsoft Developer Network (MSDN).

Properties and methods of the Embeddable Designer

In Microsoft Visual Basic, the properties of the Embeddable Designer are Read/Write. In Microsoft Visual C++, the corresponding functions will have Set and Get prefixed to the property name. Set writes to the property, and Get reads from the property. Methods and events have the same names in both environments.

Note: Only the properties and methods of the CCRDesignerCtrl class (main class for the Embeddable Designer) are included. Information on properties, methods, and events for the base classes can be found in the Microsoft Developer Network (MSDN).

Microsoft Visual Basic example

```
Dim bShowGrid As Boolean

' Get the value of the DisplayGrid property
bShowGrid = CRDesignerCtrl1.DisplayGrid

' If the Grid is displayed Set the DisplayGrid
' property to False
If bShowGrid Then CRDesignerCtrl1.DisplayGrid = False
```

Microsoft Visual C++ example

```
BOOL bShowGrid;

// Get the value for the DisplayGrid property
bShowGrid = m_Designer.GetDisplayGrid();

// If the Grid is displayed Set the DisplayGrid
// property to False
if(bShowGrid) m_Designer.SetDisplayGrid(FALSE);
```


Embeddable Designer (Properties)

Property	Data type and description	Read/Write
CaseInsensitiveSQLData	Boolean. Specifies whether or not to perform a case insensitive search for SQL data.	Read / Write
ConvertDateTimeType	CRConvertDateTimeType. Gets or sets the report option that specifies the format to be converted for date / time fields.	Read / Write
ConvertNullFieldToDefault	Boolean. Specifies whether or not to convert NULL parameter fields to their default values.	Read / Write
DisplayFieldView	Boolean. Gets or sets whether or not the field view is available. The field view is the list of available fields for the report, including Database, Formula, Parameter, Group Name, Running Total, Special, and Unbound fields.	Read / Write
DisplayGrid	Boolean. Gets or sets whether or not the grid is displayed.	Read / Write
DisplayHiddenSections	Boolean. Gets or sets whether or not a section is available when hidden.	Read / Write
DisplayRulers	Boolean. Gets or sets whether or not the Ruler is available.	Read / Write
DisplayToolbar	Boolean. Gets or sets the visibility of the toolbar.	Read / Write
EnableHelp	Boolean. Gets or sets the availability of the help options. When True, the Help icon on the toolbar is enabled and the Help command is visible in the shortcut menu. When False, the Help Icon on the toolbar is disabled and the Help command is hidden in the shortcut menu.	Read / Write
EnableSnapToGrid	Boolean. Specifies whether or not to automatically align any fields you insert or to resize to the nearest grid coordinate.	Read / Write
GridSize	Single. Gets or sets the size of the grid in inches. Default value is 0.083.	Read / Write
ReportObject	ReportObject. Gets or sets the report object.	Read / Write
TranslateDosMemos	Boolean. Gets or sets the report option that indicates whether or not to translate DOS memos.	Read / Write
TranslateDosStrings	Boolean. Gets or sets the report option that indicates whether or not to translate DOS strings.	Read / Write
UseIndexForSpeed	Boolean. Gets or sets the "Use Indexes or Server for Speed" option during record selection report option.	Read / Write
VerifyOnEveryPrint	Boolean. Gets or sets the report option that indicates whether or not to verify the database every time the report is run.	Read / Write

Embeddable Designer (Methods)

SaveReport Method

Use the SaveReport method to save the report to Crystal Reports format.

Syntax

```
Sub SaveReport (ReportName as String)
```

Parameter

Parameter	Description
ReportName	Specifies the file path and name that you want to save the report to.

This chapter provides information on using active data in the Crystal Reports Development environment. Four areas are covered: the active data drivers, the Crystal Data Object, the Crystal Data Source Type Library, and the COM Data Provider. By reading these sections you will learn how to create active data reports, create recordsets, and connect to the data source through the Report Designer Component automation server.

Active Data Drivers

Modern Visual Basic applications often use advanced ActiveX components to connect to data sources. These data sources may include Data Access Objects (DAO) using the, Remote Data Objects (RDO), OLE DB providers, such as ActiveX Data Objects (ADO), COM data providers, Crystal Data Objects (CDO) and the Crystal Data Source Type Library, or the Visual Basic data controls. Using the Active Data Drivers for Crystal Reports, you can design reports for your Visual Basic applications that use these same ActiveX data sources. Active data drivers include:

- Crystal Reports database driver for Microsoft Data Access Objects (crdb_dao.dll).
- Crystal Reports database driver for ODBC (crdb_odbc.dll) for RDO.
- Crystal Reports database driver for Microsoft ActiveX Data Objects (crdb_ado.dll).
- Crystal Reports database driver for COM data provider (crdb_com.dll).
- Crystal Data Object COM DLL (Com32.dll).
- Crystal Reports database driver for Crystal Data Object (crdb_cdo.dll).

For more information on RDO, DAO, and ADO, refer to Microsoft documentation. For information on the Data Control, refer to your Visual Basic documentation. For information on CDO, see [“Crystal Data Object” on page 308](#). For information on the Crystal Data Source Type Library, see [“Crystal Data Source Type Library” on page 311](#).

Occasionally, you may also need to create a report when the data source is not actually available at design time. Highly dynamic data may only be available at runtime. In such cases, the Field Definitions Driver (crdb_fielddef.dll) supports the use of Data definition files, which are tab-separated text files that define the fields in a data source but not the actual data.

Normally, developing applications using the Report Designer Component requires designing and saving one or more report files in advance to be accessed by the application at runtime. This process requires that the programmer has access to the data during design time, and that the application, upon installation, also installs whatever database drivers and files are required to make sure the reports can connect to the required data.

An alternative to runtime connectivity is to save data with the report files. The data is neatly packaged and available whenever the report is requested from your custom application. However, saving data with a report increases the file size of the report, wasting disk space. Furthermore, this technique produces a static report file in which the data cannot be updated without connectivity to the database.

The Field Definitions Driver allows you to create report files at design time without specifying an actual data source. Instead, the report is based on a data definition file, an ASCII text file with place holders to represent database fields. At runtime, you add code to your application to specify the actual source of data for the report.

The following topics are discussed in this section:

- “Using the Active Data Drivers” on page 299
- “Creating data definition files” on page 303
- “Using ActiveX data sources at design time” on page 305

Using the Active Data Drivers

Designing and generating reports using the Crystal Active Data Drivers is a straightforward process, but requires several specific steps:

- “Select the design-time data source” on page 299
- “Design the report” on page 300
- “Obtain a Recordset from the runtime data source” on page 300
- “Open the report” on page 301
- “Pass the Recordset to the Active Data Driver” on page 302
- “Print the report” on page 302

The following sections demonstrate this process using the Crystal Active Data Driver with the Report Designer Component Automation Server in Visual Basic 6.0.

Select the design-time data source

When designing a report for your Visual Basic application, you can specify any ActiveX data source using the Active Data Drivers, or you can specify a data definition file so that the actual data is specified at runtime only. The following example uses OLEDB (ADO) to connect to xtreme.mdb through the Microsoft Jet OLE DB provider:

- 1 Select **Using the Report Wizard** in the Crystal Reports Welcome dialog box, or click the new report icon on the Crystal Reports toolbar and select **Using the Report Wizard** in the Crystal Report Gallery dialog box.
In this example, you can click Standard from the Choose a Wizard box. Then Click OK.
- 2 In the Standard Report Creation Wizard box, under **Available Data Sources**:
 - Expand Create New Connections.
 - Expand OLE DB (ADO).
- 3 In the OLE DB (ADO) dialog box, select **Microsoft Jet 4.0 OLE DB Provider** from the Provider box and click **Next**.
- 4 In the Connection Information page:
 - Click the ellipse (...) button next to the Database Name box, browse to and select xtreme.mdb from \Program Files\Crystal Decisions\Crystal Reports 9\Samples\En\Databases.
 - Click Finish.
 In the Standard Report Creation Wizard box, under Available Data Sources and OLE DB (ADO), the selected data source is now visible.

- 5 Expand the **Tables** node and double-click **Customer** to add the table to the Selected Tables box.

Note: For information on specifying an OLE DB provider or other ActiveX data source at design time, see [“Using ActiveX data sources at design time” on page 305](#).

Design the report

Once you have selected an ActiveX data source, you can design your report just as you would design any other report.

- 1 Click **Next** to move to the Fields page.
- 2 Add fields to your report just as you would normally add fields to a report using the Standard Report Creation Wizard.
- 3 When you are finished designing the report, click **Finish**.
- 4 Apply any formatting or other changes that you feel are necessary to fine-tune the look of your report.
- 5 Save the report when finished.

Note: Before saving your report, be sure to turn off the Save Data with Report option on the File menu. The sample data stored with the data definition file is unnecessary at runtime and will only increase the size of your report file.

Obtain a Recordset from the runtime data source

Once you have selected a data source or data definition file and designed a report based on that data source or file, you can begin programming your Visual Basic application to obtain a recordset from an ActiveX data source, open the report file, set the report file to use the recordset object from the ActiveX data source, then print or export the report file. This process requires using the functionality of the Report Designer Component.

The following tutorials use the Report Designer Component Automation Server in Visual Basic 6.0. This section assumes a familiarity with the Report Designer Component Automation Server. If you need more information on how to use the automation server, see the [“Report Designer Component Object Model” on page 13](#).

To begin, you must obtain a Recordset object from a runtime ActiveX data source. This data source can be opened through DAO, RDO, ADO, the Visual Basic Data Control, Crystal Data Objects (CDO), or a class that implements the Crystal Data Source Type Library. For information on DAO, RDO, and ADO, refer to Microsoft documentation. For information on the Visual Basic Data Control, refer to your Visual Basic documentation. For information on CDO, see [“Crystal Data Object” on page 308](#). For information on the Crystal Data Source Type Library, see [“Crystal Data Source Type Library” on page 311](#).

This tutorial creates a Recordset object from the Customer table of the Xtreme.mdb sample database using ADO. The Recordset concept is used by DAO, ADO, and the Crystal Data Source Type Library. If you are using RDO, you will need to obtain a rdoResultSet object. If you are using CDO, you will need to obtain a Rowset object (see [“Crystal Data Object” on page 308](#)).

Note: You must add the ActiveX Data Object component to your Visual Basic project before performing the following steps. For instructions on using ADO with Visual Basic, refer to your Visual Basic documentation.

- Declare variables for the Database and Recordset objects in your Visual Basic application. This can be handled in the declarations section of a form or module. Use code similar to this:

```
Dim Connection As New ADODB.Connection
Dim RS As New ADODB.Recordset
```

- Create a connection to the Xtreme database.

```
Connection.ConnectionString = _
"Provider=Microsoft.Jet.OLEDB.4.0;" _
+ "Persist Security Info=False;Data Source=" _
+ "C:\Program Files\Crystal Decisions\Crystal Reports 9" _
+ "\Samples\En\Databases\xtreme.mdb;Mode=Read"
Connection.Open
```

- Obtain a Recordset object from the Customers table of the Xtreme database.

```
RS.Open "Select * From Customer", _
Connection, adOpenDynamic, adLockPessimistic, adCmdText
```

Open the report

Once you have obtained a Recordset object, you can begin working with the report file you created earlier. This example uses the Report Designer Component Automation Server to open a report file.

Note: You must add the Report Designer Component Automation Server component to your Visual Basic project before performing the following steps.

- Declare variables for the Application and Report objects that you will obtain from the Report Designer Component Object Library in the automation server. This can be handled in the declarations section of a form or module.

```
Dim CRXApplication As New Craxdrt.Application
Dim CRXReport As Craxdrt.Report
```

- Obtain a Report object by opening the report file you created earlier. This example uses the file Customer.RPT.

```
Set CRXReport = CRXApplication.OpenReport("c:\reports\Customer.rpt", 1)
```

Pass the Recordset to the Active Data Driver

The Recordset object gets passed to the Active Data Driver through the SetDataSource method of the Database object in the Report Designer Component Object Library. You must first obtain a Database object from the Report object, then you must use the SetDataSource method to set the report to point at the recordset object for your Active data source.

The following code demonstrates how to obtain a Database object from the Report object:

```
Dim CRXDatabase As Craxdrt.Database
Set CRXDatabase = CRXReport.Database
```

Once you have a Database object for the Report object, you can pass the Active data source to the Report object using the SetDataSource method. This method requires three parameters. The first is the data source itself. The second parameter is a value indicating that the data source you are passing to the report is an ActiveX data source. This value must be 3. The third parameter is the table you are passing the data source to. Since you should only have one table defining the structure of the recordset, this should always be 1. For example:

```
CRXDatabase.SetDataSource RS, 3, 1
```

Print the report

Now that the data source for the report has been set to the ADO Recordset, you can print, preview, or export the report normally. For instance, the following code prints the report to the default printer:

```
CRXReport.PrintOut
```

Once the data source has been set in the report object, runtime reporting can proceed normally. All features of the Report Designer Component are available to you. See [“Report Designer Component Object Model” on page 13](#) for more information.

Data definition files

A report file designed using a data definition file, instead of a specific database or ODBC data source, contains information about the kind of data to place in the report instead of information about an actual data source. It looks for field types, rather than actual fields.

At design time, you create your report based on the data definition file. Previewing or printing the report at design time has little value except to format field placement and style. Since there is no real data in the text file, you cannot preview or print any data at design time.

Note: You can add sample data to the data definition file so that values will appear for each field in the Preview Tab at design time, but the values will be identical for all records, and grouping will not be available.

At runtime, your application opens the report file, just as it would any other report file. Instead of simply formatting and printing the file at runtime, though, you change the data source pointed at by the Field Definitions Driver, which is the data definition file, to a Recordset or Rowset object for an ActiveX data source such as ADO, RDO, DAO, or the Crystal Data Sources (see [“Crystal Data Object” on page 308](#)), and the Crystal Data Source Type Library (see [“Crystal Data Source Type Library” on page 311](#)).

Once the Recordset from the runtime data source is passed to the report the Crystal Query engine will call the correct database driver based on the type of recordset. The Report Designer Component can then generate the actual report using the existing data. The entire process saves you time designing reports and produces reports that are much more flexible and portable at runtime. For more information on data definition files, see [“Creating data definition files” on page 303](#).

Creating data definition files

A data definition file is a tab-separated text file that contains information about field names, field types, and sample field data. Field names used in the data definition file must match the field names that will appear in the ActiveX data source that is specified at runtime. Field type information indicates the type of data in each field (string, numeric, date, and so on.) and, if it is a string field, the maximum length of the string. Finally, sample field data is simply sample data that Crystal Reports can display in the preview window while you design the report.

The following is an example of how fields are defined in a data definition file:

```
Order ID      Long      1
Customer Name String    50      Sample string value
Order Date    Date      Jan 5, 2000
Order Amount  Currency  $1.00
```

The Field Definitions Driver supports the following data types in a data definition file:

Data Type	Description
BLOB	Fields that contain bitmap images.
Boolean	True/False Boolean value.
Byte	8-bit integer value.
Currency	64-bit floating-point value that can include a currency or percent sign.
Date	Any date/time value. Examples include: <ul style="list-style-type: none"> Jan 5, 1999 07/11/97 5:06:07 07/11/97 23:30:01
Long, int32	32-bit integer value.

Data Type	Description
Memo	Any string value over 254 characters long. You must indicate the maximum number of characters for the string.
Number	64-bit floating-point value.
Short, int16	16-bit integer value.
String	Any string value under 254 characters long, such as a name, description, or identification number that is not meant to be interpreted numerically. You must indicate the maximum number of characters for the string.

Note: The data type BLOB is supported when connecting to RDO, ADO, DAO, and the data control at runtime but not when connecting to CDO.

Although data definition files can be created manually using a text editor such as Notepad, Crystal Reports provides a tool for simplifying the process. See the next section for more information.

Database Definition Tool

The Database Definition Tool is available from the Field Definitions Only dialog box when you begin designing a report based on the Field Definitions Driver. This tool allows you to design a data definition file as the first step of designing your report.

- 1 In the **Standard Report Creation Wizard**, expand **Create New Connections**, and then expand **Field Definitions**.
- 2 In the Field Definitions Only dialog box, click **Create File** to open the Database Definition Tool.
- 3 Use the Database Definition Tool to create fields for your data definition file. Use the controls to enter field names, field types, and sample data that will appear in the Crystal Reports Preview Tab. If you select String as the field type, you will also be asked to specify a maximum string length.
- 4 Click **Add** to add each new field to your data definition file. Each field appears in the list box at the bottom of the Database Definition Tool.
- 5 Continue adding as many fields as necessary for your data definition file by entering the information in the controls of the Database Definition Tool, and clicking **Add** each time. You can delete a field that you have created by selecting the field in the list box and clicking Delete.
- 6 Click **File** and then click **Save** when you are finished designing your data definition file. The Save File As dialog box appears.
- 7 Save the data definition file where it can be accessed by your report file. Click Finish and the new data definition file will appear in the Available Data Sources box under the Field Definitions Only node.

- 8 Double-click the table to add it to the **Selected Tables** box and continue creating your report.

Using ActiveX data sources at design time

The Active Data Drivers are intended to allow reports to be based on ActiveX data sources such as ADO and DAO. Data definition files allow you to avoid specifying an actual data source until runtime. However, you may often need to specify an ADO data source at design time for the report.

Each data source provides a unique dialog box to help you connect to the ActiveX data source. The dialog boxes provide options for selecting a data source to use in your report: specify an ODBC data source for RDO, specify an OLE DB provider and connection string for ADO, specify a database for DAO, specify a Prog ID for a Crystal Data Object, specify a Prog ID for a COM Data Object, or specify a data definition file. The Data Definition option has been thoroughly discussed earlier in this section. The remainder of this section will discuss selecting an ADO, RDO, DAO, CDO or COM data source.

The following topics are discussed in this section:

- “ODBC (RDO)” on page 305
- “OLE DB (ADO)” on page 306
- “Access/Excel (DAO)” on page 306
- “COM connectivity” on page 307
- “Crystal Data Object” on page 307

ODBC (RDO)

- 1 In the **Available Data Sources** box expand the **Create New Connections** node and then expand the **ODBC (RDO)** node.

The ODBC (RDO) dialog box allows you to connect to an ODBC data source. You can select from a list of available data sources, browse for a File DSN, or enter a connection string.

- Click Select Data Source to select from a list of available data sources.
 - Click Next if the ODBC data source requires log on information; specify a user name and password to log on.
- or -
- Click Find File DSN to browse to a DSN file.
 - Click Next if the ODBC data source requires log on information; specify a user name and password to log on.
- or -
- Click Connection String to add a connection string.
 - Click Next if the ODBC data source requires log on information; specify a user name and password to log on.

- 2 After you select your data source, click **Finish** to return to the Standard Report Creation Wizard.
- 3 Click **Next** in the Select Data Source dialog box.
The Select Recordset dialog box appears.
- 4 Under **ODBC (RDO)** you can now select from the **Tables**, **View**, or **Stored Procedure** node.
Note: The data source will determine which of these are available.
- 5 Continue creating your report normally.

OLE DB (ADO)

- 1 In the Standard Report Creation Wizard, expand **Create New Connections**, and then expand **OLE DB (ADO)**.
The OLE DB (ADO) dialog box allows you to select an OLE DB provider or search for a Microsoft Data Link File.
 - Click the desired OLE DB provider in the Provider box.
 - or -
 - Click Use Data Link File and then browse to a Microsoft Data Link File and select it.
- 2 Click **Next**.
This will bring you to the Connection Information page where you can enter the connection information for your data source.
- 3 Click **Finish** to return to the Standard Report Creation Wizard dialog box.
- 4 Under **OLE DB (ADO)**, you can now select from the **Tables**, **View**, or **Stored Procedure** node.
Note: The data source will determine which of these are available.
- 5 Continue creating your report normally.

Access/Excel (DAO)

- 1 In the Standard Report Creation Wizard, expand **Create New Connections**, and then expand **Access/Excel (DAO)**.
The Access/Excel (DAO) dialog box allows you to select from a list of database types including Access, dBASE, Excel, HTML, Lotus, Paradox, and Text.
- 2 Select the database type from the **Database Type** list.
- 3 Click the Ellipse button (...) for the **Database Name** box to browse and select the desired data source.

- 4 Click **Secure Logon** and enter any connection info in the boxes below if your data source requires security.
- 5 Click **Finish** to return to the Standard Report Creation Wizard dialog box.
- 6 Under **Access/Excel (DAO)** you can now select from the **Tables**, **View**, or **Stored Procedure** node.
Note: The data source will determine which of these are available.
- 7 Continue creating your report normally.

COM connectivity

- 1 In the Standard Report Creation Wizard, expand **Create New Connections**, and then expand **COM Connectivity**.
The COM Connectivity dialog box allows you to connect to a registered COM data provider on your machine. The COM data provider enumerates a list of methods on the COM object that return ADO recordsets and presents them as Stored Procedures to the Crystal Reports Designer.
- 2 Type the Prog ID in the **Program ID** box.
For example: COMDemo.SalesData
- 3 Click **Finish** to return to the Standard Report Creation Wizard dialog box.
- 4 Under **COM Connectivity**, you can now select from the list of methods provided by the COM data provider.
- 5 Continue creating your report normally.

Crystal Data Object

- 1 In the Standard Report Creation Wizard, expand **Create New Connections**, and then expand **Crystal Data Object**.
The Crystal Data Object dialog box allows you to connect to a registered Crystal Data Object provider on your machine.
- 2 Type the Prog ID in the **Program ID** box.
For example: MydataSourcePrj.MyDataSource
- 3 Click **Finish** to return to the Standard Report Creation Wizard dialog box.
- 4 Under **Crystal Data Object**, you can now select the **Crystal Data Object provider**.
Note: The data source will determine which of these are available.
- 5 Continue creating your report normally.

Crystal Data Object

The Crystal Data Object (CDO) is an ActiveX data source that allows you to define fields and records at runtime based on data that exists only at runtime. Through CDO, any data can become a virtual database and can be reported on using the power of the Report Designer Component. The Crystal Data Object does not support Memo or Blob fields.

CDO, like DAO and ADO, is based on the Component Object Model (COM). Any development environment that supports COM interfaces can dynamically generate a set of data for a report without relying on a database that exists at design time.

Applications that produce data that does not exist outside of the running application have been unable, until now, to take advantage of the most powerful reporting features in the industry. CDO, however, solves that problem. For instance, applications that monitor system or network resources, or any constantly operating environment, can produce a current report on such information at any time. No longer does data need to be dumped to a separate database before analysis. Through CDO, the Active Data Driver, and the Report Designer Component, analysis is instant and up-to-date.

The following topics are discussed in this section:

- [“CDO vs. the Crystal Data Source Type Library” on page 308](#)
- [“Using the Crystal Data Object” on page 309](#)
- [“Crystal Data Object Model” on page 311](#)

CDO vs. the Crystal Data Source Type Library

Crystal Reports also supports the [“Crystal Data Source Type Library”](#) for implementing in a Visual Basic class definition. Crystal Data Source objects can also be passed to the Active Data Driver as ActiveX data sources. However, the Crystal Data Source Type Library exposes a complete COM interface that must be implemented in your class. CDO, on the other hand, provides a fast and simple method for producing an internal customized ActiveX data source.

If you need to implement a complete data source in your application that allows runtime movement through records and fields, or if you intend to implement your data source as a separate ActiveX component, consider using the Crystal Data Source Type Library. However, if you need to create a quick and simple means of storing a large amount of data in a convenient package for reporting on, and the data will remain inside the same application as the reporting functionality, then use Crystal Data Objects.

Using the Crystal Data Object

The Crystal Data Object is an ActiveX DLL that can be accessed from any Windows development environment that supports ActiveX. By creating a Rowset object, similar to a Recordset, and filling it with fields and data, you design a virtual database table that can be passed as an ActiveX data source to the Crystal Data Object driver. The Crystal Data Object does not support Memo or Blob fields.

Once the CDO Rowset has been created, it can be used just like any other active data source such as DAO or ADO. Use a procedure, much like the procedure described in [“Using the Active Data Drivers” on page 299](#), to print, preview, or export a report at runtime that is based on the CDO data source. Simply replace the steps that explain how to pass a ADO Recordset to the Active Data Driver with appropriate steps for passing your CDO Rowset.

The rest of this section explains how to create a CDO Rowset in Visual Basic. However, as an ActiveX DLL, CDO can be used by any application development environment that supports ActiveX.

To create a CDO Rowset:

- [“Obtain a CDO Rowset object” on page 309](#)
- [“Add Fields to the Rowset object” on page 309](#)
- [“Obtain data as rows” on page 310](#)
- [“Add rows to the Rowset object” on page 311](#)

Use these steps as a guideline for creating your own CDO Rowsets for use with the Active Data Driver.

Obtain a CDO Rowset object

As stated earlier, CDO is a standard automation server. A Rowset object can be obtained from CDO using the Visual Basic CreateObject function:

```
Public CDOSet As Object
Set CDOSet = CreateObject("CrystalDataObject.CrystalComObject")
```

This Rowset object is, essentially, equivalent to a Recordset object you might obtain from DAO or another active data source. It is the Rowset object that you eventually pass to the Active Data Driver.

Add Fields to the Rowset object

Once you have a Rowset object, you need to define fields for the Rowset. These fields act as the virtual database fields. The field names you specify must match the field names specified in the data definition file. For more information on data definition files, see [“Creating data definition files” on page 303](#).

Fields are added to a CDO Rowset using the AddField method:

```
CDOSet.AddField "Order ID", vbString
CDOSet.AddField "Company Name", vbString
CDOSet.AddField "Order Date", vbDate
CDOSet.AddField "Order Amount", vbCurrency
```

This code adds four fields to the Rowset with the specified field names and field types. The field types are based on constant values for the Variant data type. The constant names used here are from Visual Basic. For information on valid constant values, see the AddField method in the [“Crystal Data Source Object Models” on page 327](#).

Obtain data as rows

Data to be added as rows in the Rowset can be collected in a two-dimensional array. The first dimension indicates rows, while the second dimension specifies fields for each row. The number of possible fields indicated by the second dimension must not exceed the number of fields you added to the Rowset using the AddField method. For example, you might define an array such as this:

```
Dim Rows(11, 3) As Variant
```

This specifies an array named Rows that contains 12 rows (0 to 11) and 4 columns (0 to 3). Notice that the four fields are defined with the AddField method, so the 4 columns in the Rows array are also defined. In addition, room has been made for 12 rows or records. Finally, since each field holds a different type of data, the array is defined as a Variant type.

Note: If your Rowset contains only a single field, you can use a one-dimensional array instead of two dimensional. The single dimension indicates the number of columns or fields in your Rowset.

Now that you have defined an array to hold data, you can begin adding values to the array. These array values will become the actual field values for the virtual database. Most likely, you will want to design a routine in your application that adds runtime data generated by your application into each cell of the array. The following code, however, demonstrates how you can explicitly add values to the array:

```
Rows(0, 0) = "1002" 'The first Order ID
Rows(0, 1) = "Cyclist's Trail Co." 'The first Company Name
Rows(0, 2) = #12/2/94# 'The first Order Date
Rows(0, 3) = 5060.2725 'The first Order Amount
```

From here, you could continue by adding a value to the first field of the second record, Rows(1, 0). You continue filling in data record by record and field by field. This technique, of course, requires a lot of code and is not very practical. Most real applications would contain a looping procedure that progressively filled in values for the array.

Add rows to the Rowset object

At this point, you have created a CDO Rowset object, added fields to the Rowset, and collected data in an array that will become part of a virtual runtime database. All that is left is to pass the data from the array to the Rowset object. This step is handled with a single method:

```
CDOSet.AddRows Rows
```

The AddRows method accepts a two-dimensional array containing the values you want added to the Rowset and, ultimately, added to a report file that is printed or exported. A one-dimensional array is used to add a single row with multiple fields.

Rows can be added to a CDO Rowset with multiple calls to the AddRows method. However, once you begin adding rows of data to a Rowset, you cannot add any new fields to the Rowset. Any call to AddFields after a successful call to AddRows will fail.

Once you finish populating your virtual database in the CDO Rowset object, you can pass this object as an active data source to the Active Data Driver using the SetDataSource method in the Report Designer Component Automation Server. For complete instructions on doing this, see [“Pass the Recordset to the Active Data Driver” on page 302](#).

Crystal Data Object Model

Crystal Data Objects support several methods and properties that can be used to work with the Rowset object. The object model for CDO is completely defined and described in the section [“Crystal Data Source Object Models” on page 327](#).

Crystal Data Source Type Library

The Crystal Data Source Type Library, like Crystal Data Objects, provides a means for designing customized data sources that can be reported off of using the Active Data Driver. Crystal Data Source, however, unlike CDO, is a type library with an interface that can be implemented in a standard Visual Basic class. Once implemented, the Crystal Data Source interface allows your data to be fully manipulated much like a standard Recordset object in ADO or DAO.

Note: The Crystal Data Source type library is designed for Visual Basic 5.0 or later.

If you simply need a quick means for packaging some data in a form that can easily be reported off of, you should consider using Crystal Data Objects. Crystal Data Source, on the other hand, is designed for developers who need more flexibility when working with custom data sources. Keep in mind, though, once you add the Crystal Data Source interface to your class, you must implement all methods and properties exposed by the interface.

The following topics are discussed in this section:

- “Creating a new project and class” on page 312
- “Adding the type library” on page 314
- “Implementing the functions” on page 316
- “Passing the CRDataSource object to the Active Data driver” on page 318
- “Crystal Data Source projects” on page 320

Creating a new project and class

The Crystal Data Source interface can be implemented inside almost any type of application. You might want to create an internal data source, for instance, inside the same standard executable application that you are implementing the Report Designer Component. On the other hand, you could create an ActiveX DLL that did nothing except implement Crystal Data Source. Your ActiveX DLL then could work as a separate data source to be accessed from other applications, much like ADO, RDO, and DAO are used.

The following topics are discussed in this section:

- “When to use the Crystal Data Source Type Library” on page 312
- “Creating a new project” on page 312
- “Adding a class module to a project” on page 313
- “Adding a Sub Main() procedure” on page 313

When to use the Crystal Data Source Type Library

The Crystal Data Source interface, as stated before, is designed to allow developers to create full-fledged data sources that work much like the ADO Recordset object. In fact, the interface has been designed to support properties and methods with names identical to several corresponding properties and methods in the ADO Recordset object. Through your existing knowledge of ADO, you can quickly familiarize yourself with the Crystal Data Source interface.

If you are designing an application or component that must produce a fully featured data source with methods and properties for easily navigating through records and fields, Crystal Data Source is the ideal solution. Not only is the interface easy to learn and use, it also follows a Recordset standard currently being developed by Microsoft.

Creating a new project

For this tutorial, you will implement the Crystal Data Source interface in an ActiveX DLL that can be referenced by other applications. One such application may be a standard executable that uses the Crystal Data Object driver with the Report Designer Component to produce reports based on this new ActiveX data source.

- 1 With Visual Basic running, select **New Project** from the **File** menu.
The New Project dialog box appears.
- 2 Select **ActiveX DLL** from the New Project dialog box, and click **OK**.
Your new ActiveX DLL project is created.
- 3 Select **Class1** in the Project window, and make sure the **Properties** window is displayed.
To display the Properties window, press the F4 key or select Properties Window from the View menu.
Note: If you are not creating an ActiveX DLL, you may not have a class module in your project. See the next section, Adding a class module to a project.
- 4 Change the value of the (Name) property for Class1 to MyDataSource.
- 5 Select **Project1** in the Project window, and change the value of the (Name) property for Project1 to MyDataSourcePrj.
- 6 Save the project.
Use MyDataSource as the name of the class file and the project file.

Adding a class module to a project

Since you are creating an ActiveX DLL, your project already contains a class module that can be used to implement the Crystal Data Source interface. However, if you are creating a project that does not automatically include a class module, such as a Standard EXE project, you will need to use the following steps.

- 1 From the Project menu, select **Add Class Module**.
The Add Class Module dialog box appears.
- 2 Make sure **Class Module** is selected, and click **Open**.
The new class module is added to your project, and the code window for the module appears.

Adding a Sub Main() procedure

Although a Sub Main() procedure is not required by ActiveX DLLs created in Visual Basic 5.0 or later, you may want to create a Sub Main() procedure to handle initialization processes. Developers working in Visual Basic 4.0 are required to add the Main subroutine to an Automation Server DLL project and specify that the project use Sub Main() as the entry point. If you are creating an ActiveX EXE in Visual Basic 4.0 or later, you should add the Sub Main() procedure to allow your code to determine if it is being started as a stand-alone application or as an out-of-process automation server.

The following steps demonstrate how to add a Sub Main() procedure in Visual Basic versions 5.0 and 6.0. If you add this procedure to the MyDataSource project, you can leave the procedure empty.

- 1 From the **Project** menu in Visual Basic, select **Add Module**.
The New Module dialog box appears.
- 2 Leave the default Module type selected, and click **Open**.
A new module, Module1, is added to your project.
- 3 In the code window for the new module, add the following code:

```
Sub Main()  
End Sub
```

Adding the type library

The Crystal Data Source interface is a standard COM (Component Object Model) interface that is defined in a type library (TLB) file. To implement the Crystal Data Source interface in your Visual Basic application, you must first add a reference to the type library, implement the interface in your class module with the *Implements* statement, and, finally, create code for each of the properties and methods defined by the interface.

The following topics are discussed in this section:

- “Adding a reference to the Crystal Data Source Type Library” on page 314
- “Viewing in the Object Browser” on page 315
- “Using Implements in the class declaration” on page 315

Adding a reference to the Crystal Data Source Type Library

If this is the first time you are using the Crystal Data Source type library, you may need to tell Visual Basic where the type library is located before you can add a reference.

- 1 From the Project menu, choose **References**.
The References dialog box appears.
- 2 Scroll through the **Available References** list to locate the **CRDataSource 1.0 Type Library**.
If you find the reference, skip to step 6. Otherwise, continue with the next step.
- 3 In the References dialog box, click the **Browse** button to locate the type library file.
The Add Reference dialog box appears.
- 4 Locate the **CRSOURCE.TLB** type library file in the same directory that you installed Crystal Reports.
If you accepted the default directory when you installed the product, this directory will be \Program Files\Crystal Decisions\Crystal Reports 9\Developer Files\include.

- 5 Select **CRSOURCE.TLB**, and click **Open**.
"CRDataSource 1.0 Type Library" will now appear in the Available References list in the References dialog box.
- 6 Select the check box next to **CRDataSource 1.0 Type Library** if it is not already selected.
- 7 Click **OK** to add the reference to your project.

Viewing in the Object Browser

Before continuing with the design of your ActiveX DLL project, it may be helpful to look at the object model provided by the Crystal Data Source interface.

- 1 From the View menu, select **Object Browser**.
The Object Browser appears.
- 2 Switch the Object Browser to display just the CRDataSourceLib object library.

Notice that the Crystal Data Source interface contains a single object: CRDataSource. This object is similar to the Recordset object you would see if you added a reference to the Microsoft ActiveX Data Objects Recordset 2.0 Library to your project. This is also the object you would pass to the Active Data Driver (Page 306) when producing a report at runtime.

Take a moment to review the properties and methods provided by the CRDataSource object. Close the Object Browser when finished.

Using Implements in the class declaration

The next step is to add the Crystal Data Source interface to your class module.

- 1 With the code window for the MyDataSource class module open, add the following code to the General Declarations section of the class:
`Implements CRDataSourceLib.CRDataSource`
- 2 Open the drop-down list of objects in the upper left of the code window.
You will see a new object has been added to the list: CRDataSource.
- 3 Select **CRDataSource** from the list of objects.
A new Property Get procedure is added to the class module for the FieldCount property of the CRDataSource object. Remember that in COM interfaces, properties are actually implemented as Get and Let procedures. For more information, refer to your Visual Basic documentation.
- 4 Open the drop-down list in the upper right of the class module code window.
Notice that several procedures appear corresponding to the properties and methods of the Crystal Data Source interface. In fact, the properties and methods you saw in the Object Browser are the same properties and methods listed here in the code window.

Implementing the functions

Once you have added the Crystal Data Source interface to your class module, you must implement all of the properties and methods in the interface to successfully produce a data source that can be compiled into an ActiveX DLL and used with the Active Data Driver. The first step to implementing all of the properties and methods is to add procedures to your class for each of the Crystal Data Source procedures.

The following topics are discussed in this section:

- [“Adding procedures” on page 316](#)
- [“Implementing procedures” on page 316](#)
- [“Compiling the ActiveX DLL” on page 317](#)

Adding procedures

When you selected the CRDataSource object in the object list in the previous section, you automatically added a procedure to the class for the FieldCount property. This property procedure appears in bold in the list of CRDataSource methods and properties to indicate that it has already been added.

- 1 With the CRDataSource object selected in the code window, select **Bookmark [Property Get]** from the drop-down list in the upper right corner of the code window.
A Property Get procedure appears in the class for the Bookmark property of CRDataSource.
- 2 Repeat the process for the Property Let procedure of the Bookmark property. Keep in mind that Property Get procedures allow values to be retrieved from properties while Property Let procedures allow values to be assigned to properties.
- 3 Continue selecting each of the property and method procedures listed so that a procedure appears in your class for every property and every method defined by the Crystal Data Source interface.
Notice that each of the procedures has been defined as *Private*. For our ActiveX DLL to expose these properties and methods to other applications, you need to change these to *Public*.
- 4 Replace each *Private* statement with *Public*.
- 5 Save your project to preserve all changes up to this point.

Implementing procedures

Exactly how you implement each of the properties and methods in the CRDataSource interface depends upon the purpose and design of your application or component. To give you an idea of how to implement the procedures, though, the following code sample simply uses an ADO Recordset object connected to the Xtreme.meb data source. Obviously, this example has little value in a real application; an ADO Recordset can itself be reported on through the OLE DB (ADO)

driver. However, the example does illustrate how the properties and methods in the Crystal Data Source interface work.

```
Implements CRDataSourceLib.CRDataSource
Dim adoRs As New ADODB.Recordset
Private Sub Class_Initialize()
Set adoRs = New ADODB.Recordset
    adoRs.Open "Customer", "Xtreme Sample Database 9", _
        adOpenKeyset, adLockOptimistic, adCmdTableEnd Sub
Private Sub Class_Terminate()
    adoRs.Close
    Set adoRs = Nothing
End Sub
Public Property Let CRDataSource_Bookmark(ByVal RHS As Variant)
    adoRs.Bookmark = RHS
End Property
Public Property Get CRDataSource_Bookmark() As Variant
    CRDataSource_Bookmark = adoRs.Bookmark
End Property
Public Property Get CRDataSource_EOF() As Boolean
    CRDataSource_EOF = adoRs.EOF
End Property
Public Property Get CRDataSource_FieldCount() As Integer
    CRDataSource_FieldCount = adoRs.Fields.Count
End Property
Public Property Get CRDataSource_FieldName _
    (ByVal FieldIndex As Integer) As String
    CRDataSource_FieldName = adoRs.Fields(FieldIndex).Name
End Property
Public Property Get CRDataSource_FieldType _
    (ByVal FieldIndex As Integer) As Integer
    CRDataSource_FieldType = adoRs.Fields(FieldIndex).Type
End Property
Public Property Get CRDataSource_FieldValue _
    (ByVal FieldIndex As Integer) As Variant
    CRDataSource_FieldValue = adoRs.Fields(FieldIndex).Value
End Property
Public Sub CRDataSource_MoveFirst()
    adoRs.MoveFirst
End Sub
Public Sub CRDataSource_MoveNext()
    adoRs.MoveNext
End Sub
Private Property Get CRDataSource_RecordCount() As Long
    CRDataSource_RecordCount = adoRs.RecordCount
End Property
```

Compiling the ActiveX DLL

Once you have finished implementing all of the properties and methods, you can compile the ActiveX DLL. When compiling ActiveX components, Visual Basic registers the component in the Windows Registry database. The name of the project,

MyDataSourcePrj in this case, is used as the name of the component. The name of the class module, MyDataSource for this example, becomes the name of a creatable object. Once compiled, the component can be referenced by another application.

- 1 Make sure you save the entire project so that all source code is preserved.
- 2 From the **File** menu, choose **Make MyDataSource.dll**.
Note: The name of the DLL that will be created is based on the name of your Visual Basic project file (.VBP), not on the project name as specified by the (Name) property.
- 3 When the Make Project dialog box appears, select the location where the new DLL should reside.
- 4 Click **OK**.
The new DLL is created and registered.

Passing the CRDataSource object to the Active Data driver

Using an object that implements the Crystal Data Source interface is a straightforward process, much like using any ActiveX component in an application. A Reference to the component must first be made, then an instance of the component object must be created in the application, and finally, the properties and methods of the object can be used. In this example, you will use the ActiveX DLL you created to obtain a MyDataSource object that can be passed to the Active Data Driver in a report generated using the Crystal Designer Component.

For this example, assume you have created an application in Visual Basic and designed a report using the Crystal Report Designer Component.

If you want to create a new report that can use the MyDataSource ActiveX DLL, create the report using three fields corresponding to the *Customer ID*, *Customer Credit ID*, and *Last Year's Sales* fields in the Customer table of the *Xtreme sample data* ODBC data source. To make things simple, you can use ADO to connect directly to those three fields. The purpose of this tutorial is simply to teach the techniques, not, necessarily, to produce a real application.

Adding a reference to MyDataSourcePrj

- 1 Choose **References** from the **Project** menu in Visual Basic.
The References dialog box appears.
- 2 Scroll through the list of **Available References** to locate the MyDataSourcePrj component.
- 3 Select the check box next to **MyDataSourcePrj**, and click **OK**.
The component is now available to your application.

- 4 Open the Object Browser in Visual Basic, and select the **MyDataSourcePrj** library.

Notice that the MyDataSource object is available and that this object contains all of the properties and methods that you implemented in the MyDataSource ActiveX DLL. Additionally, each of these properties and methods corresponds to a property or method in CRDataSource.

Creating an instance of MyDataSource

This section assumes you are already familiar with how to pass a new data source to the Active Data Driver at runtime. If you need more information on using the Active Data Driver, refer to “[Active Data Drivers](#)” on page 298. The following steps simply illustrate how to assign the myDs object created above to the Active Data Driver so that a report will use it as the source of data at runtime.

To actually use the MyDataSourcePrj component, you must create an instance of the MyDataSource object, then assign that object to the Report object displayed by your application. Assuming you created a report in your application using the Crystal Designer Component and accepted default settings for adding the Crystal Report Viewer/ActiveX to your project:

- 1 Open the code window for the form containing the CrystalReport Viewer/ActiveX.
- 2 In the General Declarations section for the form, add the following code:
- 3 In the Form_Load procedure, add the following line before the Report object is assigned to the ReportSource property of the CRViewer1 object:

```
Dim myDs As New MyDataSourcePrj.MyDataSource
```

```
CRXReport.Database.SetDataSource myDs,3, 1
```

Note: This example is based on a Visual Basic application created using the Report Designer Component. for more information see “[Report Designer Component Object Model](#)” on page 13.

The first line of code creates an instance of the MyDataSource object in your application, much like you might create an instance of an ADO Recordset object. The second line of code added uses the SetDataSource method inside the Crystal Designer Component library to dynamically change the source of data used by your report.

If you designed your report using an ADO, DAO, or RDO data source, or by using a Data Definition file, then your report uses the Active Data Driver to access the report data at runtime. Since the Active Data Driver also supports data sources that expose the Crystal Data Source interface, you can easily assign the MyDataSource object to your report.

Crystal Data Source projects

Now that you have seen the extensive power of the Crystal Data Source interface implemented inside a Visual Basic class, you can begin to consider the extensive possibilities for its use. Many computer-based operations produce continuous streams of data in real time. In your own work, you may encounter needs for gathering data from process control systems, data acquisition applications, or computer-based instrumentation.

In these kinds of applications, data is usually gathered and stored for later analysis. Systems running in real time, however, may need real-time monitoring and reporting. With objects that implement the Crystal Data Source interface, you can gather and move through data as it is generated, then produce up to the instant analysis through reports.

Programmer's building n-tier applications that operate across a network may often find themselves designing business objects and other business rules components. By implementing the Crystal Data Source interface in business object components, you can design reports that produce real-time information about data traveling across the network. Even Microsoft Transaction Server components can implement a fully functional ActiveX data source for reporting. Crystal Data Source takes your applications from runtime to real time.

COM data provider

The Crystal Reports database driver for COM data provider (crdb_com.dll) allows Crystal Reports to report off COM data. Based on a given ProgID, this driver will enumerate the list of methods on the COM object that return ADO recordsets and will present them as Stored Procedures to the Crystal Reports Designer. The methods on the object can require parameters that will also propagate back to the Crystal Reports Designer, which will treat them as a standard Crystal Reports Parameter Type. This driver also supports accessing Chaptered or Hierarchical ADO recordsets.

Creating a COM Data provider is a straightforward process. This tutorial demonstrates how to create a COM Data provider that returns a recordset from the Xtreme Sample Database 9 ODBC data source, and how to connect to that data source.

Tip: The Xtreme Sample Database 9 ODBC data source is installed by default. If you do not have this data source installed, create the data source and point it to `\Program Files\Crystal Decisions\Crystal Reports 9\Samples\En\Databases\xtreme.mdb`

The following topics are discussed in this section:

- [“Advantages of the COM data provider” on page 321](#)
- [“Creating a new project and class” on page 321](#)
- [“Creating a Report off the COM Data driver” on page 325](#)

Advantages of the COM data provider

The COM data provider provides significant enhancements to developers including:

- The ability to report off of and manage multiple data sources.
Traditionally, reports created in Crystal Reports have been created from physical databases with the option of connecting to a database using either a native driver or an ODBC driver. With the new COM driver, Crystal Reports is available to other types of data sources. You can now create a report so that it can report from Active Data in the report designer.

- The ability to easily separate report design from application creation when working with Active Data record sets.

With the new COM Driver, the act of connecting to the data source and generating the recordset from which the report can be designed are now separate. You no longer need to write code to push data into the report. This means report designers can simply point to the DLL, and create their report without any concerns about having adequate sample data, access to the actual data, and so on. It is much simpler than designing reports via TTX information and one line of sample data. Unlike Data Definition (TTX) files, this method is for use only during design time. You should be able to use these reports at runtime; however, you cannot pass a recordset to the report at runtime (as can be done with TTX files). Another advantage of using this method is the ability to put your own interface into the report designer.

Creating a new project and class

The Crystal Data Source interface can be implemented inside almost any type of application. You might want to create an internal data source, for instance, inside the same standard executable application that you are implementing the Report Designer Component. On the other hand, you could create an ActiveX DLL that did nothing except implement Crystal Data Source. Your ActiveX DLL then could work as a separate data source to be accessed from other applications, much like ADO, RDO, and, DAO are used.

The following topics are discussed in this section:

- “Creating a new project” on page 322
- “Adding a class module to a project” on page 322
- “Adding a Sub Main() procedure” on page 323
- “Adding a reference to the Microsoft ADO Library” on page 323
- “Creating the functions” on page 323
- “Compiling the ActiveX DLL” on page 324

Creating a new project

For this tutorial, you will create an ActiveX DLL that can be referenced by other applications. One such application may be a standard executable that uses the COM Data driver with the Report Designer Component to produce reports based on this new ActiveX data source. The code will create functions that allow the user to select all the tables records, the records from a specific region, or the records with sales over a specific amount.

- 1 With Visual Basic running, select **New Project** from the **File** menu.
The New Project dialog box appears.
- 2 Select **ActiveX DLL** from the New Project dialog box, and click **OK**.
Your new ActiveX DLL project is created.
- 3 Select **Class1** in the Project window, and make sure the **Properties** window is displayed.
To display the Properties window, press the F4 key or select Properties Window from the View menu.
Note: If you are not creating an ActiveX DLL, you may not have a class module in your project. See the next section, Adding a class module to a project.
- 4 Change the value of the (Name) property for Class1 to DemoXtreme.
- 5 Select **Project1** in the Project window, and change the value of the (Name) property for Project1 to DemoCom.
- 6 Save the project.

Use DemoXtreme as the name of the class file and DemoCom as the name of the project file.

Adding a class module to a project

Since you are creating an ActiveX DLL, your project already contains a class module. However, if you are creating a project that does not automatically include a class module, such as a Standard EXE project, you will need to complete the following steps.

- 1 From the **Project** menu, select **Add Class Module**.
The Add Class Module dialog box appears.
- 2 Make sure **Class Module** is selected, and click **Open**.
The new class module is added to your project, and the code window for the module appears.

Adding a Sub Main() procedure

Although a Sub Main() procedure is not required by ActiveX DLLs created in Visual Basic 5.0 or later, you may want to create a Sub Main() procedure to handle initialization processes. Developers working in Visual Basic 4.0 are required to add the Main subroutine to an Automation Server DLL project and specify that the project use Sub Main() as the entry point. If you are creating an ActiveX EXE in Visual Basic 4.0 or later, you should add the Sub Main() procedure to allow your code to determine if it is being started as a stand-alone application or as an out-of-process automation server.

The following steps demonstrate how to add a Sub Main() procedure in Visual Basic versions 5.0 and 6.0. If you add this procedure to the MyDataSource project, you can leave the procedure empty.

- 1 From the **Project** menu in Visual Basic, select **Add Module**.
The New Module dialog box appears.
- 2 Leave the default Module type selected, and click **Open**.
A new module, Module1, is added to your project.
- 3 In the code window for the new module, add the following code:

```
Sub Main()  
End Sub
```

Adding a reference to the Microsoft ADO Library

You need to reference the ADO library in order to create and pass the recordset to the report. To add the reference:

- 1 From the **Project** menu, choose **References**.
The References dialog box appears.
- 2 Scroll through the Available References list to locate the **Microsoft ActiveX Data Object 2.x library**.
- 3 Select the check box next to **Microsoft ActiveX Data Object 2.x library**.
- 4 Click **OK** to save.

Creating the functions

Once you have added the ADO reference to your project, you can add the code to create and pass the recordset.

```
Dim Connection As New ADODB.Connection  
Dim adoRs As New ADODB.Recordset
```

```
Private Sub Class_Initialize()  
Set adoRs = New ADODB.Recordset  
adoRs.Open "Customer", "Xtreme Sample Database 9", _  
adOpenKeyset, adLockOptimistic, adCmdTableEnd Sub
```

```

Public Function demoSelectRegion(ByVal Region As String) As ADODB.Recordset
Dim sqlStm As String
sqlStm = "SELECT * FROM Customer WHERE Region = '" _
& sRegion & "'"
Set demoSelectRegion = Connection.Execute(sqlStm)
End Function

Public Function demoSelectSales(ByVal Sales As Double) As ADODB.Recordset
Dim sqlStm As String
sqlStm = "SELECT * FROM Customer WHERE `Last Year's Sales` > " _
& Sales
Set demoSelectSales = Connection.Execute(sqlStm)
End Function

Public Function demoRecordset() As ADODB.Recordset
Dim sqlStm As String
sqlStm = "SELECT * FROM Customer "
Set demoNoParam = Connection.Execute(sqlStm)
End Function

Private Sub Class_Terminate()
cnn.Close
End Sub

```

Note: It is the responsibility of the Provider to close any recordsets that may be open.

Compiling the ActiveX DLL

Once you have finished adding the functions, you can compile the ActiveX DLL. When compiling ActiveX components, Visual Basic registers the component in the Windows Registry database. The name of the project, DemoCom in this case, is used as the name of the component. The name of the class module, DemoXtreme for this example, becomes the name of a creatable object. Once compiled, the component can be referenced by the COM Data driver.

- 1 Make sure you save the entire project so that all source code is preserved.
- 2 From the **File** menu, choose **Make DemoCom.dll**.
Note: The name of the DLL that will be created is based on the name of your Visual Basic project file (.VBP), not on the project name as specified by the (Name) property.
- 3 When the Make Project dialog box appears, select the location where the new DLL should reside.
- 4 Click **OK**.
The new DLL is created and registered.

Creating a Report off the COM Data driver

Once you have created the COM Data provider, you can create a report off the new data source.

- 1 Select **Using the Report Wizard** in the Crystal Reports Welcome dialog box, or click the new report icon on the Crystal Reports toolbar and select **Using the Report Wizard** in the Crystal Report Gallery dialog box.
In this example, you can click Standard from the Choose a Wizardbox. Then Click OK.

- 2 In the Standard Report Creation Wizard box, under **Available Data Sources**:
 - Expand Create New Connections.
 - Expand COM Connectivity.

- 3 In the COM Connectivity page:
 - Type DemoCom.DemoXtreme in the Program ID box.
 - Click Finish.

In the Standard Report Creation Wizard box, under the COM Connectivity node, in the Available Data Sources box, the selected data source is now visible.

- 4 Double-click **demoRecordset** to add the Stored Procedure from the Com Data provider to the Selected Table box.
- 5 Click **Next** to move to the Fields page.
- 6 Add fields to your report just as you would normally add fields to a report using the Standard Report Creation Wizard.
- 7 When you are finished designing the report, click **Finish**.
- 8 Apply any formatting or other changes that you feel are necessary to fine-tune the look of your report.
- 9 Save the report when finished.

Crystal Reports provides support for reporting off data when no true data source exists. In this chapter you will find detailed information, including properties and methods, on Crystal Data Objects and the Crystal Data Source Type Library.

Crystal Data Source Object Models

Crystal Reports includes two ActiveX based data source models to allow on-the-fly reporting when a true data source does not exist at design time, and the data at runtime does not exist in a relational or OLAP database. The Crystal Data Sources allow you to dynamically produce data at runtime inside your code, then pass the data to an existing report file. Both data source models are designed primarily for Visual Basic programmers, but they can be used within other development environments that support ActiveX components and interfaces.

Crystal Data Objects

Crystal Data Objects allow you to quickly design a set of relational data at runtime using standard Visual Basic arrays. For more information on using Crystal Data Objects in Visual Basic, see [“Crystal Data Object” on page 308](#). To add a reference to the Crystal Data Objects component to your Visual Basic application, select the Crystal Data Object item in the Available References list box of the References dialog box. Crystal Data Objects do not support Memo or Blob fields.

CrystalComObject

The CDO component provides a single object named CrystalComObject. This object works much like a Recordset or Rowset that you might use in ADO, DAO, or RDO. Rather than connecting to an existing database, though, CDO allows you to fill it with data stored in a standard Visual Basic array. Once filled with data, the entire object can be passed to the [“Grid Controls and the Crystal Report Engine”](#) at runtime, producing a dynamic report filled with data only available at runtime.

To create an instance of the CrystalComObject in Visual Basic, use the following code as an example:

```
Dim cdoRowset As Object
Set cdoRowset = CreateObject("CrystalComObject.CrystalDataObject")
```

The following topics are discussed in this section:

- [“CrystalComObject Properties” on page 329](#)
- [“CrystalComObject Methods” on page 329](#)

CrystalComObject Properties

The CrystalComObject provides a single property:

RowCount

Use this property to obtain the number of rows in the rowset once data has been assigned. This is especially useful if data is added to the Rowset in several steps, each step adding more to the size of the Rowset. This value can be used to find out how many rows have been added.

Example

```
Dim numRows As Long
numRows = cdoRowset.RowCount
```

CrystalComObject Methods

The CrystalComObject uses the following methods. These methods each have a section describing their parameters and returns, followed by an example.

- “AddField” on page 329
- “AddRows” on page 330
- “DeleteField” on page 331
- “GetColCount” on page 331
- “getEOF” on page 331
- “GetFieldData” on page 332
- “GetFieldName” on page 332
- “GetFieldType” on page 333
- “MoveFirst” on page 333
- “MoveNext” on page 334
- “MoveTo” on page 334
- “Reset” on page 334

AddField

```
Function AddField(FieldName As String, [FieldType]) As Boolean
```

Use this method to add fields to a rowset before adding data. The rowset must have fields defined before data can be added using the AddRows method.

Parameters

FieldName

A string value specifying the name of the field.

FieldType

An optional value specifying the data type that will be contained in this field. Use Visual Basic VarType constants to specify the data type. If this value is omitted, the vbVariant type will be used.

Returns

A Boolean value indicating whether or not the field was successfully added to the Rowset.

Example

```
cdoRowset.AddField "Order ID", vbString  
cdoRowset.AddField "Company Name", vbString  
cdoRowset.AddField "Order Amount", vbCurrency
```

AddRows

```
Sub AddRows(RowData)
```

Use this method to assign an array of data to the CDO Rowset.

Parameters

RowData

A standard Visual Basic two-dimensional array. The first dimension specifies the number of rows in the Rowset, while the second dimension specifies the number of fields for each row. A one-dimensional array is used to add a single row with multiple fields. When this array is dimensioned, it must be defined As Variant. For example:

```
Dim Rows(11, 3) As Variant
```

This example creates an array that will hold 12 rows with 4 fields. You must assign data to all cells in the array before assigning the array to the Rowset using AddRows.

Example

```
cdoRowset.AddRows Rows
```

DeleteField

Function DeleteField(FieldName As String) As Boolean

This method removes an existing field from the Rowset. If the field contains any data, that data is lost.

Parameters

FieldName

The name of the field you want to delete from the Rowset.

Returns

A Boolean value indicating whether or not the field was successfully deleted. If the field does not exist, this function will return False.

Example

```
cdoRowset.DeleteField "Company Name"
```

GetColCount

Function GetColCount() As Integer

This function returns the number of columns or fields currently in the Rowset.

Returns

An integer value indicating the number of fields in the Rowset.

Example

```
Dim numFields As Integer
numFields = cdoRowset.GetColCount
```

getEOF

Function getEOF() As Boolean

Use this function to determine if the current row in the Rowset is the last row.

Returns

True if the current row is the last row in the Rowset. False if the current row is anywhere else in the Rowset.

Example

```
If cdoRowset.getEOF Then
    cdoRowset.MoveFirst
End If
```

GetFieldData

Function GetFieldData(column As Integer)

This method obtains the current value of a specific column in the Rowset for the currently selected row.

Parameters

column

A number indicating which field (column) of the row you want the current value of. Rowset columns, like array dimensions, are 0 based. The first column is 0, the second is 1, and so on.

Returns

A variant value that contains the data for the specified field of the current row.

Example

```
Dim fieldData As Variant  
fieldData = cdoRowset.GetFieldData 0
```

GetFieldName

Function GetFieldName(column As Integer) As String

Returns the name of the specified field (column). Field names are assigned using "AddField".

Parameters

column

A number indicating which field (column) of the Rowset you want the name of. Rowset columns, like array dimensions, are 0 based. The first column is 0, the second is 1, and so on.

Returns

A string containing the name of the specified field.

Example

```
Dim secondField As String  
secondField = cdoRowset.GetFieldName 1
```

GetFieldType

Function GetFieldType(Field) As Integer

Use this method to obtain the type of data contained by a field in the Rowset. Data types are assigned to fields using “[AddField](#)”.

Parameter

Field

This parameter indicates which field you are querying for the type of data contained. This field can accept either a numeric value or a string value. If you use a numeric value, it must be a number representing the field (column) of the Rowset you want to find out the data type of. Rowset columns, like array dimensions, are 0 based. The first column is 0, the second is 1, and so on. If a string is used, the string must contain the name of the field.

Returns

A Visual Basic VarType constant indicating the type of data contained in the field.

Example

```
Dim dataType As Integer
dataType = cdoRowset.GetFieldType "Customer Id"
If dataType = vbString Then
    ' Do something with the string
End If
```

MoveFirst

Function MoveFirst() As Boolean

This method moves to the first row (record) in the Rowset.

Returns

A Boolean value indicating whether or not the current row was successfully set to the first row. If the Rowset contains no data, this method will return False.

Example

```
cdoRowset.MoveFirst
```

MoveNext

Function MoveNext() As Boolean

Moves to the next row (record) in the Rowset. The current record is set to the new row.

Returns

A Boolean value indicating whether or not the current record was successfully set to the next row in the Rowset. This function will return False if the current record before calling the method is the last row of the Rowset.

Example

```
cdoRowset.MoveNext
```

MoveTo

Function MoveTo(recordNum As Long) As Boolean

Moves the current record to the specified record number in the Rowset.

Parameter

recordNum

A 1-based value indicating to which record in the Rowset you want to move. The first record is 1, the second is 2, and so on.

Returns

A Boolean value indicating whether or not the current record was successfully set to the specified record number. This method returns False if the specified record does not exist.

Example

```
cdoRowset.MoveTo 9
```

Reset

Sub Reset()

Resets the Rowset, clearing all fields and data.

Example

```
cdoRowset.Reset
```


Crystal Data Source Type Library

The Crystal Data Source Type Library is a COM interface type library that can be implemented in your own Visual Basic classes. Once the interface has been added to a class, you must implement every method and property defined by the interface. This process requires extensive Visual Basic coding, but the result is a complete data source that can be used much like other ActiveX data sources such as ADO or DAO.

Possible uses for data sources defined using the Crystal Data Source Type Library are ActiveX style data sources, similar to ADO, DAO, and RDO, Business Objects and business rules components, Microsoft Transaction Server components, or instrumentation systems that produce dynamic real-time data.

Note that as a type library, the Crystal Data Source Type Library does not actually provide any functionality on its own. You must determine the actual functionality by implementing each of the properties and methods defined in the type library interface. The descriptions given here of the CRDataSource object, its properties, and its methods are intended as a guideline for the type of functionality you should define in your own classes.

CRDataSource

CRDataSource is a COM interface rather than an actual COM object. By writing code for each of the CRDataSource methods and properties in your own code, your class or COM component implements the CRDataSource interface and, therefore, becomes the actual Crystal Data Source.

To implement CRDataSource in a Visual Basic class, use the following code in the General Declarations section of your class module:

```
Implements CRDataSourceLib.CRDataSource
```

Once implemented, CRDataSource will appear as an available object in your class module. You must add code for every property and method of the CRDataSource object to correctly implement the Crystal Data Source interface. For more information, see [“Crystal Data Source Type Library” on page 311](#).

The following topic is discussed in this section:

- [“CRDataSource Properties” on page 336](#)

CRDataSource Properties

The Crystal Data Source interface defines the following properties. An example of each property follows the property description.

- [“Bookmark” on page 336](#)
- [“EOF” on page 336](#)
- [“FieldCount” on page 336](#)
- [“FieldName” on page 337](#)
- [“FieldType” on page 337](#)
- [“FieldValue” on page 337](#)
- [“RecordCount” on page 337](#)

Bookmark

Used to obtain a bookmark (identifier) for a particular record in the Recordset, or to move to the record identified by the bookmark. To simplify the process of navigating to a particular record over and over again, you can save a bookmark for the record in a variable, then quickly return to that record by assigning the value of the variable to the Bookmark property.

Example

```
Dim aBookmark As Variant
aBookmark = myCRDataSource.CRDataSource_Bookmark
' Move around in the Recordset performing various operations
' To return to the bookmarked record:
myCRDataSource.CRDataSource_Bookmark = aBookmark
```

EOF

Read only.

This property indicates whether or not the current record is the last record in the Recordset. The value of the EOF property is True if the current record is the last record, False otherwise.

Example

```
If myCRDataSource.CRDataSource_EOF = True Then
    myCRDataSource.CRDataSource_MoveFirst
End If
```

FieldCount

Read only.

This property returns the number of fields in the Recordset.

Example

```
Dim numFields as Integer
numFields = myCRDataSource.CRDataSource_FieldCount
```

FieldName

Read only.

Returns the name of a specific field as indicated by the field index. Field indexes for the Crystal Data Source interface start with 1 for the first field, 2 for the second, and so on.

Example

```
Dim firstField As String
Dim secondField As String
firstField = myCRDataSource.CRDataSource_FieldName 1
secondField = myCRDataSource.CRDataSource_FieldName 2
```

FieldType

Read only.

Obtains a Visual Basic VarType constant indicating the type of data stored in a particular field. Fields are identified using field indexes. Field indexes for the Crystal Data Source interface start with 1 for the first field, 2 for the second, and so on.

Example

```
Dim fieldType As Integer
fieldType = myCRDataSource.CRDataSource_FieldType 1
If fieldType = vbString Then
    ' Do something with the string data in this field
End If
```

FieldValue

Obtains the data actually stored in the field for the current record. Fields are identified using field indexes. Field indexes for the Crystal Data Source interface start with 1 for the first field, 2 for the second, and so on.

Example

```
Dim fieldVal As Variant
fieldVal = myCRDataSource.CRDataSource_FieldValue 2
```

RecordCount

Read only.

This property contains the total number of records in the Crystal Data Source recordset.

Example

```
Dim numRecords as Long
numRecords = myCRDataSource.CRDataSource_RecordCount
```

CRDataSource Methods

The Crystal Data Source type library defines the following two methods:

- “MoveFirst” on page 338
- “MoveNext” on page 338

MoveFirst

```
Sub MoveFirst()
```

Moves to the first record in the Recordset. The first record is set as the current record.

Example

```
myCRDataSource.CRDataSource_MoveFirst
```

MoveNext

```
Sub MoveNext()
```

Moves to the next record in the Recordset and sets that record to the current record. A Visual Basic error occurs if the current record before calling this method is the last record. Use “EOF” to determine if the current record is the last record in the recordset.

Example

```
If Not myCRDataSource.CRDataSource_EOF Then  
    myCRDataSource.CRDataSource_MoveNext  
End If
```

Creating User-Defined Functions in C 10

Crystal Reports allows you to create User-Defined Functions that are recognized by the Crystal Reports Formula Editor. In this chapter you will find detailed information on programming User-Defined Functions in C.

Overview of User-Defined Functions in C

The Crystal Reports Formula Editor and formula language are powerful tools, enabling you to perform a wide variety of report-related tasks easily and efficiently. The formula language is expandable as well. That is, while it already includes a large selection of useful functions, it also comes with the ability to accept new functions that you define to meet your needs.

User-Defined Functions that are recognized by the Crystal Reports Formula Editor can be created in a Dynamic Link Library or in an Automation Server. This section demonstrates how to create User-Defined Functions in a Dynamic Link Library, the U25 DLL, using the C programming language.

Note:

- The U25 DLLs work with Crystal Reports version 5 or later.
- Existing U2L DLLs (the earlier version of the UFL DLL) can still be used.

For information on how to create User-Defined Functions in an Automation Server using Visual Basic, see [“Creating User-Defined Functions in Visual Basic” on page 359](#).

Programming User-Defined Functions in C

You can add new functions to the Crystal Reports Formula Editor by:

- Writing the functions using the C programming language.
- Compiling and linking the functions into a User-Defined Function DLL called a UFL (User Function Library).

Note: If you are not familiar with programming Windows DLLs, refer to the Microsoft Windows Software Development Kit. Do not attempt to create a UFL if you do not understand Windows DLL programming.

When designing a new function for the Crystal Reports Formula Editor, you need to determine the following:

- [“Name of the function” on page 340](#)
- [“Purpose of the function” on page 341](#)
- [“Function data” on page 341](#)
(The report data or user supplied data that the function will require.)
- [“Return types” on page 342](#)
(The type of data that the function will return to the report.)

Name of the function

The name of the new function should reflect the function’s purpose, making it easier to recognize when it appears in the Formula Editor’s Functions list. For example, a function named “Picture” could let you specify a template picture of

how data should appear in the report. If a field contains phone numbers, you can use the Picture function to specify that the data appear like this:

(xxx) xxx-xxxx

A resulting value from the phone number field would appear as follows:

(415) 555-1234

Function names must start with a letter, while all remaining characters in the name can be letters or numbers. The name can be up to 254 characters long, and it must be unique. That is, you cannot give a function a name that has been used for another Formula Editor function or UFL function, nor can it have a name that matches a standard C keyword (such as if, return, switch, or while).

Purpose of the function

You may find it helpful to start simply by “fleshing out” your function. Determine the purpose of the function and outline it on paper. Use C code or even pseudocode to determine how the function will perform the required operation. This step is important, as it will form the base information for every other step in the designing and programming of your UFL.

Function data

UFL functions are much like any other function you might create in C. They can accept values that are passed as parameters, and they return a value that is printed on the report. Once you have determined how a UFL function will perform a task, you will know exactly what kind of data it will require to complete that operation. The following table shows the data types that a UFL function can accept as a parameter, along with a description of what the parameter will look like in C.

Parameter Type (alphabetical)	C Data Type
Array (Boolean)	Pointer to a Boolean array.
Array (Currency)	Pointer to a number array.
Array (Date)	Pointer to a date array.
Array (DateTime)	Pointer to a date time array.
Array (Number)	Pointer to a number array.
Array(NumberToCurrency)	Pointer to a number array.
Array (String)	Pointer to a string array.
Array (Time)	Pointer to a time array.
Boolean	Short integer.
Currency	Double.
Date	Integer.
DateTime	Structure of two integers.

Parameter Type (alphabetical)	C Data Type
Number	Double.
NumberToCurrency	Double.
Range (Currency)	Structure containing two doubles.
Range (Date)	Structure containing two long integers.
Range (DateTime)	Structure containing two date time structures.
Range (Number)	Structure containing two doubles.
Range (NumberToCurrency)	Structure containing two doubles.
Range (String)	Structure containing pointers to two elements in a character array.
Range (Time)	Structure containing two integers.
RangeArray (Currency)	Pointer to a number range structure array.
RangeArray (Date)	Pointer to a date range structure array.
RangeArray (DateTime)	Pointer to a date time range structure array.
RangeArray (Number)	Pointer to a number range structure array.
RangeArray (NumberToCurrency)	Pointer to a number range structure array.
RangeArray (String)	Pointer to a string range structure array.
RangeArray (Time)	Pointer to a time range structure array.
String	Pointer to an array of characters.
Time	Integer.

Return types

Finally, you must determine what kind of data your function returns to the current report in Crystal Reports. The following table lists the possible UFL return types along with a description of the C data type used when programming the function.

Return Type (alphabetical)	C Data Type
Array (Boolean)	Pointer to a Boolean array.
Array (Currency)	Pointer to a number array.
Array (Date)	Pointer to a date array.
Array (DateTime)	Pointer to a date time array.
Array (Number)	Pointer to a number array.
Array (String)	Pointer to a string array.
Array (Time)	Pointer to a time array.
Boolean	Short integer.

Return Type (alphabetical)	C Data Type
Currency	Double.
Date	Long integer.
DateTime	Structure of two integers.
Number	Double.
Range (Currency)	Structure containing two doubles.
Range (Date)	Structure containing two long integers.
Range (DateTime)	Structure containing two date time structures.
Range (Number)	Structure containing two doubles.
Range (String)	Structure containing pointers to two elements in a character array.
Range (Time)	Structure containing two integers.
RangeArray (Currency)	Pointer to a number range structure array.
RangeArray (Date)	Pointer to a date range structure array.
RangeArray (DateTime)	Pointer to a date time range structure array.
RangeArray (Number)	Pointer to a number range structure array.
RangeArray (String)	Pointer to a string range structure array.
RangeArray (Time)	Pointer to a time range structure array.
String	Pointer to a character array.
Time	Integer.

Programming the UFL

After sketching out a new function, deciding on its parameters and data types, and determining the type of data the function will return to a report, you can begin programming the UFL.

A UFL is like any other DLL with a few simple differences:

- Although the file is referred to as a “UFL” it must have a U25 prefix. For example, U25SAMP.DLL.
- It must export your User-Defined Function (UDF) as a DLL function.
- It must export a collection of other functions required by Crystal Reports.

You can design your UFL from scratch, but you may find the “**Helper modules**”, provided with Crystal Reports useful. If you use the helper modules provided, you will only need to create a single C code module, a “**Module Definition (.def) file**”, and an application project file. More experienced programmers may want to use these modules simply as a starting point to design more complex UFL features.

The “Picture” example

Code for the Picture function is provided as an example of a UFL. The purpose of this function is to display string type field data using a format specified by the user. For example, if the phone number is entered in the database table as “4155551234”, the user can create the following formula:

```
Picture({table.PHONENUM}, “(xxx) xxx-xxxx”)
```

The phone number will appear in the report as: (415) 555-1234

The code segments that appear in this section as examples use this Picture function. In addition, the complete code is listed in “Picture function—a sample UFL function”, later in the chapter.

Helper modules

The files listed below have been installed on your system in \Program Files\Crystal Decisions\Crystal Reports 9\Developer Files\include.

You do not need to work with the actual code in any of these files, but they will have to be added to your UFL project. UFJOB.H and UFJOB.C are optional files providing job information for the current print job accessing the Formula Editor. For more information, see “UFJOB modules” on page 356.

File Name	Purpose
crdll.h	Defines primitives required by ufdll.h.
crdates.h	Declarations for functions to convert to and from Crystal Reports' date values.
crdates.c	Implementation of functions to convert to and from Crystal Reports' date values.
ufdll.h	Defines UFL enumerated, union, structure, and other data types.
ufuser.h	Prototypes of tables and functions user must implement.
ufmain.h	Prototypes of internal UFL functions implemented in UFMAIN.C.
uffuncs.h	Prototypes of functions that must be exported by the UFL to be used by Crystal Reports. These functions are implemented in UFMAIN.C.
ufmain.c	Implementation of UFL functions used internally and used by Crystal Reports when connecting to the UFL. This file also contains generic LibMain and WEP functions for Windows 3.x DLLs and a generic DllEntryPoint function for Win32 DLLs. (The LibMain, WEP, and DllEntryPoint functions are defined conditionally according to whether or not you are building a Win32 DLL. You do not need to make any changes to the code here).
ufjob.h	Optional file. Contains a definition of the JobInfo structure (see “UFJOB modules” on page 356, and prototypes of the “InitForJob function” on page 350, “TermForJob function” on page 350,) and FindJobInfo (see “Implement InitJob and TermJob” on page 357, implemented in UFJOB.C. structure and PROTOTYPES.C.)
ufjob.c	Optional file. Contains implementations of the required “InitForJob function” and “TermForJob function”. Also implements the FindJobInfo helper function.

Setting up a UFL project

Begin creating your UFL by setting up a new directory on your system to work in. Use File Manager or Windows Explorer to copy all of the files listed in the chart under “**Helper modules**” into your new directory. This ensures that you do not inadvertently edit or change the original files in your CRW directory.

Note: If you will not be using the functionality provided by UFJOB.H and UFJOB.C, you do not need to copy these files into your working directory.

Open Microsoft Visual Studio and create a new project file. Name the new project file with a U25 prefix. For example, U25Funcs.dsp. Make sure the project is set to build a DLL (not a Windows EXE) file.

Note: If you are building a User Function Library, make sure you set Struct Member Alignment for the project to 1 byte. Crystal Reports for Win32 will not be able to use your UFL otherwise. To find out how to change the Struct Member Alignment setting for your project, refer to the documentation for your development environment.

Finally, add the UFMain.c file to your project, and save the project in your new working directory. Add the UFJOB.C file as well if you are using this file. You are now ready to begin creating a new UFL for the Crystal Reports Formula Editor.

UFL states

To keep a report section together during an operation (for instance, formatting) Crystal Reports sometimes needs to roll back to the last consistent point—perhaps the end of the last section that fits completely on a page. Crystal Reports accomplishes this by saving and then restoring state information, including UFL states. A UFL state consists of the global information maintained by the UFL and modified by its functions. The UFL generates its state as a binary stream. An example is a UFL that implements a running total. It would have the running total, a function to reset the total to zero, another to increment it, and a third to retrieve its current value. That running total is the UFL’s state and the first two functions mutate it. For the Crystal Reports Print Engine to produce consistent results, the UFL must implement the save and restore functions, UF5SaveState and UF5RestoreState. To save the UFL state at particular points in time, Crystal Reports calls UF5SaveState to obtain the current state from the UFL. Later, Crystal Reports calls UF5RestoreState to restore the UFL state to its saved values.

Note: As the developer, it is your responsibility to ensure that the type of state information saved can be restored correctly. For example, pointers representing memory addresses should not be part of the UFL state data. The UFL state is serialized and stored in a report so if that report is opened on another machine, the pointers will be invalid. You should also consider whether the report will be opened on different platforms. In this case, the UFL functions that save and restore its state should be platform-aware. For example, an integer is stored by a Pentium III using little-endian representation, while the same integer is stored by a

a SPARC using big-endian representation. It is the UFL's responsibility to interpret the meaning of the binary stream and to insure its integrity: the UFL must ensure that little-endian order is always used, regardless of the platform's native byte order.

Function definition

Creating a UFL function requires that you create only one more C code module and a module definition (.def) file in addition to the Helper modules. (Your particular UFL Function needs may require more modules, but the simplest method for creating a UFL requires creating only these two.) For information on creating the module definition file, see ["Module Definition \(.def\) file" on page 355](#). To begin building your UFL, create a new C module in your Visual Studio, and save it to your working directory with the same name as your project file. For example, if your project file is named U25samp.mak or U25Funcs.dsp, you would name your C module U25Samp.C or U25Funcs.C. This demonstration will refer to UFLSamp.C. This is the "private" C module for your UFL because it is the one section of code that must be unique to your UFL.

The first step to programming your UFL's private C code module is to #include the appropriate header files:

- #include <windows.h>
- #include "ufdll.h"
- #include "ufmain.h"
- #include "ufuser.h"

You do not need to #include the UFFUNCS.H header file that you also copied into your working directory. This file is already #included by UFMAIN.C, and you will not be directly calling any of the functions defined in these files (though they are necessary for Crystal Reports when the Formula Editor accesses your UFL).

The private C code module of your UFL requires several parts:

- ["Function definition table" on page 347](#)
- ["Function definition table example" on page 348](#)
- ["Function templates table" on page 348](#)
- ["Function examples table" on page 349](#)
- ["Error table" on page 349](#)
- ["InitForJob function" on page 350](#)
- ["TermForJob function" on page 350](#)
- ["UFL function implementation" on page 351](#)

Most of these sections have specific guidelines that must be used and that are the same for every function you add to your UFL. Your UFL function implementation is completely designed and programmed by you. It is the functionality that you are adding to the Crystal Reports Formula Editor.

Function definition table

You must supply a definition for each function that you want to add to Crystal Reports. Each entry in the function definition table consists of a definition string that specifies:

- The return type.
- The function name.
- An argument list (showing the required order in which arguments are to be entered and the data type of each argument).
- The name of the new UFL C/C++ function that implements the call (that is, the function for which this entry is being written).
- The name of the C/C++ function that implements the memory calculation call or NULL if this function is not available.
- A Boolean value. When this value is TRUE, the UFL tells the formula compiler that a formula that uses this UFL function should be marked automatically as a print-time formula.
- A Boolean value. When this value is TRUE, the UFL tells the formula compiler that this UFL function has side effects. In this case, the UFL must implement the UF5SaveState and UF5RestoreState functions.

The definition table entry for a UFL function is in the following format:

```
"returnType UDFName (arg1, arg2,...)", CFunctionName,
MemCalcFunctionName, BooleanValue, BooleanValue"
```

Here is an example:

```
"String Picture (String, String)", Picture, NULL, FALSE, FALSE
```

In this example:

- "String" specifies that the function is to return a string.
- "Picture" is the name that will identify the function on the Function list in the Formula Editor.
- "(String, String)" specifies that the function is to require two arguments, both are strings.
- "Picture" (appearing after the comma), is the actual function name, the name you give the function when you code it.
- "NULL" indicates there is no memory calculation function available.
- "FALSE" indicates that the function is not called only while printing records.
- "FALSE" indicates that the function does not has side effects.

Note: The UDFName and the CFunctionName do not have to be the same. You can use something other than the function name to identify a function on the Function list of the Formula Editor if you wish.

All function definitions must be set up in a table with the following heading:

```
UF5FunctionDefStrings FunctionDefStrings [] =
```

Note: The entry must be terminated with three NULLs and two FALSEs.

Crystal Reports uses the information you supply in this table to create parameter blocks when you call the functions.

Function definition table example

Here is a sample function definition table entry for the Picture function:

```
UFFFunctionDefStrings FunctionDefStrings [] =
{
    {"String Picture(String,String)",Picture, NULL, FALSE, FALSE},
    {NULL,NULL,NULL, FALSE, FALSE}
};
```

Here, the third argument being NULL indicates that no memory calculation function is used (Crystal Reports will use the maximum allowable). The last two arguments being FALSE indicate the Picture function is not print-time only and has no side effects.

In contrast, if a memory calculation function is used by the Picture function, the definition might be:

```
{"String Picture (String, String)", Picture, PictureRetSize, FALSE, FALSE},
```

Here, the PictureRetSize function will calculate the maximum length of the string returned by the Picture function.

Function templates table

You must supply a function template for each function that you define. A function template specifies the string that Crystal Reports is to enter in the Formula Editor's Formula text box when you select the function from the Functions list in the Formula Editor. Each template must contain:

- The function call.
- Any syntax guides (parentheses, commas, etc.) you want to include.
- An exclamation point (!) character to specify where the insertion point is to appear when the function is entered into the formula. Here is an example:

```
"Picture (!, )"
```

In this example, the string "Picture (,)" is to be entered into the formula whenever you select the Picture function from the Functions list in the Formula Editor.

- The string includes the function call "Picture" and the syntax guides "(,)" to guide the user when entering arguments.
- The exclamation point specifies that the insertion point is to be placed inside the parentheses, before the comma.

All function templates must be set up in a table with the following heading:

```
UFFFunctionTemplates FunctionTemplates [] =
```

Note: The table must be terminated with a null.

Continuing with the example, the function templates table for the Picture function should look like this:

```
UFFunctionTemplates FunctionTemplates [] =
{
    {"Picture (!, )"},
    {NULL}
};
```

Function examples table

You must also supply a function example for each function. A function example specifies the string you want to use to identify and select the function in the Functions list in the Formula Editor. Here is a function example for the Picture function:

```
"\tPicture (string, picture)"
```

This example specifies that the string "Picture (string, picture)" is the listing that you want to appear in the Functions list. The characters "\t" specify that the string is to be set in from the left one tab stop, thus aligning it with other functions in the list.

All function examples must be set up in a table with the following heading:

```
UFFunctionExamples FunctionExamples [] =
```

Note: The table must be terminated with a null.

The complete function examples table for the Picture function should look like this:

```
UFFunctionExamples FunctionExamples [] =
{
    {"\tPicture (string,picture)"},
    {NULL}
};
```

Error table

You must also supply an error string for each error message that you want to make available to your functions. Each error string contains only the text you want to display when an error is detected, and it must be in the format:

```
"ErrorString"
```

Note: C compilers view the first string in the table as Error 0, the second as Error 1, and so on. Each string in the error table corresponds to an error code that you define. For each user error code, there must be a message string in the error table at the corresponding index, where the first string is at index 0.

Each UFL C function must return a value of the enumerated type `UFL_Error`, defined in `ufdll.h`. Return `UF_NoError` if no error occurred. Return one of the other `UFL_Error` values if there is an error. Try to choose one of the predefined values if it fits your situation. If the error is specific to your UFL, set the `Return_Value.UFL_Return_User_Error` member of the parameter block to an error code value you have defined and return

UFLUserError from your function code. The Formula Editor will then call back to return the error string that you have defined in the error table.

The Formula Editor passes a parameter block to a UFL function rather than individual parameters. [“Obtaining parameter values from the parameter block” on page 352](#), will examine how to handle parameter blocks.

All error strings must be set up in a table with the following heading:

```
char *ErrorTable [] =
```

An error table for the Picture function should, at a minimum, look like this:

```
char *ErrorTable[] =
{
    “no error”
};
```

InitForJob function

The InitForJob function initializes all user function UFL's with the job ID whenever a job starts. You can handle any job initialization for your functions here, but all that is absolutely necessary is an empty function implementation:

```
void InitForJob(UFTInt32u jobID)
{
}
```

Note: See the note following [“TermForJob function” on page 350](#).

TermForJob function

The TermForJob function terminates the job ID for all user function UFL's whenever a job finishes. You can add any job termination code here that you like, but all that is absolutely necessary is an empty function implementation:

```
void TermForJob(UFTInt32u jobID)
{
}
```

Note: Every UFL must have an implementation of [“InitForJob function”](#), and TermForJob. These functions are called when a job starts and ends printing, respectively. You can choose to implement these yourself. At a minimum, you must provide empty functions. Alternatively, if you include the UFJOB helper modules in your project, you do not need to implement InitForJob and TermForJob. Crystal Reports provides helper modules (UFJOB.C and UFJOB.H) which implement these functions and maintain a doubly linked list of JobInfo structures, one for each active job. The JobInfo structure (declared in UFJOB.H) holds the ID# of the job and contains a void pointer where you can store any data that you allocate. The files also implement a FindJobInfo function (see [“Implement InitJob and TermJob” on page 357](#)), which you can use to retrieve the job information for any open job. [“UFJOB modules”](#) will examine these files and their implementation of InitForJob and TermForJob.

UFL function implementation

As the final step to creating a UFL function (see “[Function definition](#)” on page 346), you must add code for the operation of the function you have designed. Your function must be programmed for the specific needs of your UFL. This section will examine the basics of how to obtain the parameters from the parameter block and use the values of those parameters in the implementation of the UFL function.

You begin by coding your function as follows:

```
ExternC UError FAR _export PASCAL FunctionName
    (UParamBlock *ParamBlock)
{
    // Your function's code
}
```

First, notice that the function is exported because a UFL is simply a DLL being accessed by Crystal Reports. Second, the function returns an error code of the enumerated type `UError`. In the error trapping sections of your function, you can return a `UError` value, such as `UFNotEnoughParameters`, or you can return `UFUserError` and define your own errors in the error table. Finally, the function accepts a parameter block of type `UParamBlock` from Crystal Reports rather than individual parameters. You will need to retrieve the individual parameters from that parameter block to work with the data values passed by the Formula Editor.

Note: `ExternC` is defined in `UFDLL.H` and is equivalent to: `extern “C”`.

Returning user-defined errors

The `UError` enumerated type provides several errors that are common to many types of functions. If appropriate, have your UFL function return one of these predefined types. If no error occurs, you can return `UFNoError`.

If your function can cause an error that is not predefined, you can establish a user-defined error. You do this by:

- Adding an error string to the error table.
- Passing the appropriate error index to the `ReturnValue.UFReturnUserError` member of the parameter block.
- Returning `UFUserError` from your UFL function implementation.

A user-defined error string in the error table might look like this:

```
char *ErrorTable[] =
{
    “My User-defined Error”
};
```

This error string is assigned an error index by Crystal Reports. The first error is 0, the second is 1, etc. If an error occurs in your function, you assign the appropriate index value to the `ReturnValue.UFReturnUserError` member of the parameter block. For example:

```
ParamBlock->ReturnValue.UFReturnUserError = 0;
```

Once you specify a user-defined error, you can return the `UFError` value `UFUserError` from your function. When the Formula Editor finds the error in a formula entered in the Formula text box (when the Check or Accept button is clicked), it will use the error index specified to report the appropriate string listed in the error table.

Obtaining parameter values from the parameter block

Since Crystal Reports passes the values for a Formula Editor function's parameters in a parameter block rather than individually, you must separate the values from the parameter block before they can be evaluated. For the `Picture` function (see ["Picture function—a sample UFL function" on page 353](#)), expect the parameter block to contain two parameters, both of type `String`, as defined in the function definition table.

To obtain the values of the individual parameters, begin by defining pointers to two structures of type `UFParamListElement` (defined in `UFDLL.H`):

```
UFParamListElement*FirstParam,*SecondParam;
```

Next, call the `GetParam` function (defined in `UFMAIN.C`) for each `UFParamListElement` to obtain a pointer to each parameter from the parameter block:

```
FirstParam = GetParam (ParamBlock, 1);  
SecondParam = GetParam (ParamBlock, 2);
```

The actual value is stored in the `Parameter` member of the `UFParamListElement` according to the type of data it contains. The `Parameter` member is a `UFParamUnion` type (a union data type holding a value according to the type of data expected). Since both of the parameters are of type `String`, you can obtain the actual parameter value for each `UFParamListElement` by using the following notation:

```
FirstParam->Parameter.ParamString  
SecondParam->Parameter.ParamString
```

If, on the other hand, the second parameter contained a numeric value, you could use this notation:

```
SecondParam->Parameter.ParamNumber
```

Study the `UFParamUnion` union definition in the `UFDLL.H` file for a complete list of all possible parameter types and how to obtain a value from them.

Picture function—a sample UFL function

Here is a complete commented private C code module implementing the Picture function. Use this as a guide for how to create your own functions in UFLs:

```

/*****
** U25SAMP.C
**
** Private C code module
** implementing the Picture UDF.
*****/

#include <Windows.h>
#include "UFD11.h"
#include "UFMain.h"
#include "UFUser.h"

#define PicturePlaceholder 'x'

/* UDF PROTOTYPE */
ExternC UError FAR PASCAL Picture    (UFParamBlock * ParamBlock);

/*****
* This array gives the program the types for the
* parameters to the UDF, the return
* type, and the name. It also passes the
* address of the actual function code.
*****/

UF5FunctionDefStrings FunctionDefStrings [] =
{
    {"String Picture (String, String)", Picture, NULL, FALSE, FALSE},
    {NULL, NULL, NULL, FALSE, FALSE}
};

/*****
* The following is the template the program
* will insert into the Formula Editor
* Formula text box when this function is
* selected.
*****/

UFFunctionTemplates FunctionTemplates [] =
{
    {"Picture (!,)",
     {NULL}}
};

/*****
* The following is an example of the format
* for this function. This text will appear

```

```
* in the Functions list box of the Formula
* Editor.
*****/

UFFunctionExamples FunctionExamples [] =
{
    {"\tPicture (string, picture)"},
    {NULL}
};

/*****
* This array contains ASCII string
* representations of the errors which
* could occur.
*****/

char *ErrorTable [] =
{
    "no error"
};

/* Called for on initialization */
void InitForJob (UFTInt32u jobID)
{
}

/* Called on termination */
void TermForJob (UFTInt32u jobID)
{
}

/*****
* This function is used by the Picture UDF to
* copy the contents of a source string and a
* format string into a destination string.
*****/

static void copyUsingPicture(char *dest, const char *source, const char
*picture)
{
    while (*picture != '\0')
    {
        if (tolower (*picture) ==
            PicturePlaceholder)
            if (*source != '\0')
                *dest++ = *source++;
            else
                ; // do not insert anything
            else
                *dest++ = *picture;
            picture++;
    }
    // copy the rest of the source
```

```

    lstrcpy (dest, source);
}

/*****
*   This is the User-Defined Function
*****/

ExternC UError FAR PASCAL Picture (UParamBlock * ParamBlock)
{
    UParamListElement*FirstParam,*SecondParam;
    FirstParam = GetParam (ParamBlock, 1);
    SecondParam = GetParam (ParamBlock, 2);

    if (FirstParam == NULL || SecondParam == NULL)
        return UFNotEnoughParameters;
    copyUsingPicture(ParamBlock->ReturnValue.ReturnString,
        FirstParam->Parameter.ParamString,
        SecondParam->Parameter.ParamString);
    return ULError;
}

```

Module Definition (.def) file

The last element of your UFL is the module definition (.def) file. This is just like any module definition file you would create for a DLL, but you must make sure to explicitly export not only your UFL function, but also the specialized UFL functions defined in UFMMAIN.C that Crystal Reports expects to find. The following is an example of a module definition file for the UFL that exports the Picture function (see [“Picture function—a sample UFL function” on page 353](#)):

```

LIBRARY      U25SAMP
DESCRIPTION 'Sample User Function Library for Crystal Reports'
EXPORTS
    UF5Initialize
    UF5Terminate
    UF5GetVersion
    UF5StartJob
    UF5EndJob
    UF5GetFunctionDefStrings
    UF5GetFunctionExamples
    UF5GetFunctionTemplates
    UF5ErrorRecovery
    Picture

```

Notice that the only function exported that you actually coded is the Picture UFL. The rest of the exported functions have been defined for you in UFMMAIN.C. Every U25 DLL must export these 9 UF* functions. If UF5SaveState and UF5RestoreState are defined, they should also be exported.

When you have finished coding the module definition file, save it to your working directory and add it to the list of files in your project file.

Finally, compile and link the u25*project file. Resolve any errors that occur, and recompile if necessary. Once you have your DLL (u25*.dll), place it in \Program Files\Common Files\Crystal Decisions\2.0\bin directory. From that point on, when you open the Formula Editor, your User-Defined Function(s) will appear in the “Additional Functions” section at the bottom of the Functions list of the Formula Editor. Enter each function in one or more formulas, and test and modify it until it works the way you want.

Note: For additional information, review UFLSamp1.c (a sample file that was installed in \Program Files\Crystal Decisions\Crystal Reports\Developer Files\include).

UFJOB modules

Two optional modules, UFJOB.C and UFJOB.H have been provided with Crystal Reports. These files provide an implementation of the “InitForJob function”, and “TermForJob function”, which allow you to obtain an ID number that is specific to the current print job in Crystal Reports. At the same time, these modules establish a JobInfo structure for the current job where you can store information regarding the job. If your UFL, for example, must evaluate all values in a field before printing a result, it can tally data in the JobInfo structure until it has a result. Data can even be passed between functions using the JobInfo structure.

Use the JobInfo structure whenever you want to create UFL functions that summarize or group report data. For example, statistical functions that evaluate the median, mean, or range of values in a field can store data in the JobInfo structure.

If you decide to use the UFJOB modules in your own UFL, the following are required:

- “UFJOB.C” on page 356
 - Add UFJOB.C to your UFL project file.
- “UFJOB.H” on page 357
 - #include UFJOB.H in your private C code module that implements your UFL functions.
- “Implement InitJob and TermJob” on page 357
 - Replace your own implementations of the InitForJob and TermForJob functions with implementations of the InitJob and TermJob functions.

UFJOB.C

This module contains implementations of the “InitForJob function” on page 350, and “TermForJob function” on page 350, that provide a JobInfo structure for the current job in Crystal Reports. These replace any versions of these functions that you coded in your private C module containing your UFL definitions. They also call the InitJob and TermJob functions respectively. You will implement these functions in your private C code module.

In addition, this file also defines the FindJobInfo function. Use this function in your own code whenever you need to obtain a pointer to the JobInfo structure for the current job. This function requires only the job ID number, which is stored in the JobId member of the parameter block. The following code demonstrates how to call this function:

```
struct JobInfo *jobInfo;
jobInfo = FindJobInfo (ParamBlock->JobId);
```

UFJOB.H

This module prototypes the “InitForJob function”, “TermForJob function”, and FindJobInfo functions that appear in UFJOB.C. It also prototypes the InitJob and TermJob functions that you must implement yourself. Most importantly, this file defines the JobInfo structure:

```
struct JobInfo
{
    UFTInt32u jobId;
    struct JobInfo FAR *prev;
    struct JobInfo FAR *next;

    void FAR *data;
};
```

The data member of this structure allows you to store a pointer to any kind of data you wish. This means you can store information for the current job that allows you to evaluate several field values before printing a result or store data from one function to be used by another function.

Implement InitJob and TermJob

If you previously implemented the InitForJob and TermForJob functions in the private C code module with your UFL function definitions, delete those functions now since they are being replaced by the functions in UFJOB.C. Now, in your private C module, define the InitJob and TermJob functions which InitForJob and TermForJob call. You can use these functions to add job initialization or job termination code to your UFL, but they can also remain blank.

Following are examples of how you might implement these functions for your own UFL:

```
void InitJob (struct JobInfo *jobInfo)
{
}

void TermJob (struct JobInfo *jobInfo)
{
    If (jobInfo->data != 0)
        free (jobInfo->data);
}
```

Once you have done all this, you can make full use of the JobInfo structure with your own UFL function code.

Special functions

You can define several special purpose functions in your Class Module in addition to the User-Defined Functions that you are creating. The Crystal Reports Formula Editor can look for and process code defined in any of the following functions:

Be sure to define the functions in your code exactly as they appear above. If not defined correctly, they will be ignored by Crystal Reports.

UF5Initialize

This function is called just after the DLL is loaded into memory. Use this function to handle one-time initialization of variables. It takes a version number.

UF5Initialize returns a value of 0 (zero) to indicate successful initialization. Any non-zero values indicate initialization failed.

UFTerminate

This function is called just before the DLL is unloaded from memory. Use this function to clean up any allocated memory or other data before unloading the DLL.

UFTerminate returns a value of 0 (zero) when the memory clean-up is finished.

UFStartJob

This procedure is called by Crystal Reports just before User-Defined Functions are evaluated (or reevaluated). The Crystal Reports job number is passed to the function as the only parameter. Use this function to handle any initialization on a per-job basis.

UFEndJob

This procedure is called by Crystal Reports when the current job finishes (that is, when all pages of a report have been generated), before UFStartJob is called again, or before UFTerminate is called. This function also accepts the Crystal Reports job number as its only parameter. Handle any clean-up necessary on a per-job basis using this function.

UF5SaveState

This procedure must be implemented only if your UFL has a function with side effects. It is called by Crystal Reports to obtain the state from the UFL at a given point. The Crystal Reports job number is passed to the function. The function returns the UFL state binary stream in the second parameter, and returns the size of the state in the third parameter.

UF5RestoreState

This procedure must be implemented only if your UFL has a function with side effects. It is called by Crystal Reports to restore the UFL state to its saved value. The Crystal Reports job number and the saved UFL state binary stream are passed to the function.

Creating User-Defined Functions in Visual Basic

11

Crystal Reports allows you to create User-Defined Functions that are recognized by the Crystal Reports Formula Editor. In this chapter you will find detailed information on programming User-Defined Functions in Microsoft Visual Basic.

Overview of User-Defined Functions in Visual Basic

The Crystal Reports Formula Editor and formula language are powerful tools, enabling you to accomplish a wide variety of report-related tasks easily and efficiently. The formula language is expandable as well. That is, while it already includes a large selection of useful functions, it also comes with the ability to accept new functions that you define to meet your needs.

User-Defined Functions that are recognized by the Crystal Reports Formula Editor can be created in a Dynamic Link Library or, for 32-bit environments, in an Automation Server. This section demonstrates how to create User-Defined Functions in an Automation Server using Visual Basic. For information on how to create User-Defined Functions in a Dynamic Link Library using the C or C++ programming language, see [“Programming User-Defined Functions in C” on page 340](#).

Programming User-Defined Functions in Visual Basic

The Crystal Reports Formula Editor can access User-Defined Functions through a User Function Library (UFL). A User Function Library is a specially designed Dynamic Link Library that exposes one or more functions that you create. UFLs can be designed and programmed in any language that supports the development of Windows DLLs.

The Crystal Reports Professional Edition includes the User Function Library U2LCOM.DLL. This UFL is installed in your `\Program Files\Common Files\crystal decisions\2.0\bin` directory when you install Crystal Reports and provides an interface through which you can expose User-Defined Functions in Automation Servers.

An automation server is a Dynamic Link Library or executable file that exposes its functionality to other modules and processes through the Component Object Model. For complete information on the Component Object Model (COM), refer to Microsoft documentation or the Microsoft World Wide Web site. U2LCOM.DLL is a UFL that can access any functions exposed by any Automation Servers named with a CRUFL prefix. This means that you can create an Automation Server in any language environment that supports COM, name the server with a CRUFL prefix, register the server on a system, and the Crystal Reports Formula Editor will access and make available any functions exposed by that Automation Server.

Note: You may be more familiar with Automation Servers as OLE Automation. Though the technology behind Automation Servers is the same as OLE Automation, the correct name for these servers is now simply Automation Servers.

Visual Basic, version 4.0 and later, allows you to design 32-bit Automation Servers easily. As a Visual Basic programmer, you can design user-defined functions for use in your reports by exposing them in ActiveX DLLs and registering the DLL on your system.

Note: Any development environment that allows the creation of COM-based Automation Servers can use the techniques similar to those described in this section. However, the code and examples shown here are based on Visual Basic.

The steps for creating automation servers in Visual Basic are slightly different, depending on the version of Visual Basic you are using.

Using Visual Basic 4.0

There are five primary steps to creating an Automation Server in Visual Basic 4.0 that exposes functions to the U2LCOM.DLL User Function Library:

- “Set Up the Main Subroutine” on page 361
- “Add a Class Module to the project” on page 361
- “Add User Functions to the Class Module” on page 362
- “Name the project” on page 362
- “Compile the project as an OLE DLL” on page 362

Set Up the Main Subroutine

When Visual Basic 4.0 first opens, it creates a default project and form for you. The entry point to your Automation Server DLL must be the Main subroutine. This subroutine needs to be defined in a Visual Basic Module, but cannot be defined in a Class Module.

- 1 Remove the default form from your project by highlighting the form in the Project window and choosing **Remove File** from the **File** menu.
- 2 To add a new Module to the project, choose **Module** from the **Insert** menu.
- 3 In the Module window, add the following:

```
Sub Main()  
End Sub
```

You do not need to add any code to the Main subroutine, as long as it exists in your project.

Add a Class Module to the project

Now you must add a Class Module to your project. The Class Module will contain any User-Defined Functions that you wish to make available to the Crystal Reports Formula Editor.

- 1 To create a Class Module, choose **Class Module** from the **Insert** menu. A new Class Module window appears, and the Class Module is added to the Project window.
- 2 In the Properties window, set the following properties for the Class Module:
 - **Instancing** = 2 - Creatable MultiUse
 - **Name** = Any valid Class Module name
 - **Public** = True

Note: For complete information on these properties, refer to Visual Basic Help.

If the Properties window is not visible, select the Class Module window, then choose Properties from the View menu.

Add User Functions to the Class Module

The next step is to add the actual User-Defined Functions that you want to appear in the Crystal Reports Formula Editor. These functions are standard Visual Basic functions that you might define in any Visual Basic project. The only requirement is that the functions are declared Public.

Type in the following function:

```
Public Function DateToString(date1 As Date) As String
    DateToString = Format(date1, "Long Date")
End Function
```

Note: For complete information on how to define functions in Visual Basic, refer to your Visual Basic documentation.

The U2LCOM.DLL UFL supports most standard Visual Basic data types and arrays. For complete information on how the Crystal Reports Formula Editor interprets Visual Basic data types and arrays, see the sections “[Visual Basic and Crystal Reports](#)” on page 365, and “[Using Arrays](#)” on page 366.

Name the project

The U2LCOM.DLL UFL will only read Public functions exposed by Automation Servers named with a CRUFL prefix. For example, *CRUFLMyFunctions* is a valid project name for your Automation Server.

- 1 To change the name of your project in Visual Basic 4.0, choose **Options** from the **Tools** menu.
The Options dialog box appears.
- 2 Click the **Project** Tab, and change the name of the project in the **Project Name** text box.
The name should be CRUFLxxx, where xxx is a name of your choice.
- 3 While in the **Options** dialog box, make sure the Startup Form for the project is the Sub Main subroutine.
- 4 Click **OK** when finished.

Compile the project as an OLE DLL

- 1 Save the Module file, the Class Module file, and the P3project file for your project.
- 2 Choose **Make OLE DLL File** from the **File** menu.
- 3 Accept the default name for the DLL (your file name and location can be anything), and click **OK**.

Visual Basic creates your Automation Server for you and registers it in your local system Registry.

Using Visual Basic 5.0

There are only four major steps to creating an Automation Server in Visual Basic 5.0. When you finish designing your Automation Server, the U2LCOM.DLL User Function Library will be able to access all of the functions that it exposes.

- [“Create a New ActiveX DLL project” on page 363](#)
- [“Add User Functions to the Class Module” on page 363](#)
- [“Name the project” on page 363](#)
- [“Build the DLL” on page 364](#)

Create a New ActiveX DLL project

- 1 When you first run Visual Basic 5.0, the New Project dialog box appears. If you already have Visual Basic open, simply choose **New Project** from the **File** menu.
- 2 Double-click the **ActiveX DLL** icon in the New Project dialog box.
Visual Basic creates a new project for you with a single Class Module.

Note: For complete information on creating an ActiveX DLL in Visual Basic 5.0, refer to your Visual Basic documentation.

Add User Functions to the Class Module

The next step is to add the actual User-Defined Functions that you want to appear in the Crystal Reports Formula Editor. These functions are standard Visual Basic functions that you might define in any Visual Basic project. The only requirement is that the functions are declared Public.

Type in the following function:

```
Public Function DateToString(date1 As Date) As String
    DateToString = Format(date1, "Long Date")
End Function
```

The U2LCOM.DLL UFL supports most standard Visual Basic data types and arrays. For complete information on how the Crystal Reports Formula Editor interprets Visual Basic data types and arrays, see the sections [“Visual Basic and Crystal Reports” on page 365](#), and [“Using Arrays” on page 366](#).

Name the project

The U2LCOM.DLL UFL will only read Public functions exposed by Automation Servers named with a CRUFL prefix. For example, CRUFLMyFunctions is a valid project name for your Automation Server.

To change the name of your project in Visual Basic 5.0, highlight your project in the Project window, then change its (*Name*) property in the Properties window to CRUFLxxx, where xxx is a name of your choice.

Build the DLL

From the File menu, choose Make CRUFLxxx.dll. Once again, xxx is the name you chose. When this command executes, Visual Basic will compile your project into an ActiveX DLL (an Automation Server) and register it in your local system Registry.

Registration of the Automation Server and Distribution of the Visual Basic Project

An Automation Server must be registered on any system on which it will be used. Automation Servers are registered in the Windows 95 or Windows NT System Registry. The file can be physically stored anywhere on your hard drive because the Registry settings tell Windows where the file is located when it is needed.

When you make an ActiveX DLL project in Visual Basic 5.0 or an OLE DLL project in Visual Basic 4.0, using the Make command from the File menu, Visual Basic automatically registers the Automation Server on your system. Once registered, you can easily test and use the new User-Defined Functions from the Crystal Reports Formula Editor.

Visual Basic 4.0 also includes an easy-to-use command-line application, REGSVR32.EXE, that will handle registering an Automation Server on your system. This application is located in the \CLISVR subdirectory of the directory in which you installed Visual Basic. This application can be used from an MS-DOS prompt by simply specifying the name of the Automation Server DLL. For example:

```
RegSvr32.exe C:\Projects\CRUFLMyFunctions.dll
```

The easiest way to distribute any Visual Basic project is to use the Setup Wizard to create an installation program. In addition to making sure all necessary files are installed where appropriate, the Setup Wizard can add code to your installation to register the Automation Server in the user's system Registry.

In addition to installing and registering your Automation Server, you must also provide the U2LCOM.DLL UFL file on any system that will use the functions exposed by your Automation Server. If the system has Crystal Reports installed, then this file will already be located in the \Program Files\Common Files\Crystal Decisions\2.0\bin directory.

Using the User-Defined Functions

- 1 In Crystal Reports, create a new report and add tables to it, or open an existing report.
- 2 From the **Insert** menu, choose **Formula Field**.
The Insert Field dialog box appears.
- 3 Click **New**, and enter a name for the new formula in the Formula Name dialog box.
- 4 Click **OK**.
The Formula Editor appears.

- 5 Scroll down in the Functions list box to the Additional Functions section. Locate the User-Defined Function you created.
User-Defined Function names for functions created in Automation Servers are prefixed according to the project and class name used when you created the Automation Server. For instance, if you named your project *CRUFLProject*, named the class *Conversion*, and named the function *Square*, the User-Defined Function would appear in the Formula Editor as *ProjectConversionSquare*.
- 6 Double-click the User-Defined Function, and it appears in the **Formula** text box.
- 7 Enter valid arguments for the function.
For example:
`ProjectConversionSquare(5)`
- 8 Click **Check**. Make sure no errors appear in the function.
- 9 Click **Accept**, and place the formula in your report.
- 10 Preview the report and verify that the function worked correctly.
Congratulations, you just created and used a User-Defined Function.

Visual Basic and Crystal Reports

Note: 32-bit Support Only. U2LCOM.DLL is a 32-bit UFL only and, therefore, supports only 32-bit Automation Server DLLs. To create User-Defined Functions for Crystal Reports, you must have the 32-bit edition of Visual Basic 4.0 or Visual Basic 5.0 (or another 32-bit development environment that supports the creation of COM-based Automation Servers).

Variable Types

Crystal Reports will support most common Visual Basic data types provided through a User-Defined Function developed in Visual Basic. The following table shows how Crystal Reports converts the most common Visual Basic data types to data types supported by the Formula Editor:

Visual Basic Data Types	Formula Editor Data Types
Integer Long Single Double	NumberVar
Currency	CurrencyVar
Date	DateVar
Boolean	BooleanVar
String	StringVar

Note: Ranges—the range data type available in the Crystal Reports Formula Editor is not supported in COM-based User-Defined Functions.

Using Arrays

Arrays can be passed to any User-Defined Function as a parameter of the function. This means that when you design your function in Visual Basic, the function can accept an array of values of any of the supported data types. However, the function cannot return an array to the Crystal Reports Formula Editor. The following Visual Basic function is an acceptable User-Defined Function for Crystal Reports:

```
Public Function GetNthItem (sArray() As String, n As Integer) As String
    GetNthItem = sArray(n)
End Function
```

Reserved Names

Certain names are reserved and cannot be used as User-Defined Function names. The following names are reserved by the Crystal Reports Formula Editor for special purposes:

- UFInitialize
- UFTerminate
- UFStartJob
- UFEndJob

For more explanation of these function names, see [“Special purpose functions” on page 368](#). In addition, User-Defined Functions cannot use the same name as any of the functions exposed by the IDispatch interface used by COM:

- QueryInterface
- AddRef
- Release
- GetTypeInfoCount
- GetTypeInfo
- GetIDsOfNames
- Invoke

Function Name Prefixing

To ensure a unique name when User-Defined Functions appear in the Formula Editor, Crystal Reports appends a prefix to each function name that is generated from the project and Class Module names in the original source code. The first part of the prefix is the project name without the CRUFL prefix. The rest of the function name prefix is the Class Module name.

Once the prefix for the function name is generated, all non-alphanumeric characters are removed, and the prefix is applied to the original function name. The following table illustrates this process:

Project Name:	CRUFLTestFunctions
Class Module Name	Conversion
User-Defined Function Name	Date_To_String()
Formula Editor Function	TestFunctionsConversionDateToString()

This function name prefixing can be turned off if you are sure that your function names will not conflict with any other function names recognized by the Formula Editor. To turn off function name prefixing:

- 1 Define a Boolean property for the class called UFPrefixFunctions.
- 2 Set the value of the property to False in the Initialize subroutine for the class.

For example:

```
Public UFPrefixFunctions As Boolean
Private Sub Class_Initialize()
    UFPrefixFunctions = False
End Sub
```

Note: Function name prefixing is designed to eliminate function name conflicts. If you turn off function name prefixing and your function name conflicts with another function, you may get unpredictable results.

Passing Parameters By Reference and By Value

Arguments can be passed to User-Defined Functions written in Visual Basic either ByRef or ByVal. ByRef is the default method for Visual Basic, but both methods will work. For instance, all of the following functions are valid:

```
Public Function Test1 (num1 As Integer) As Integer
Public Function Test1 (ByRef num1 As Integer) As Integer
Public Function Test1 (ByVal num1 As Integer) As Integer
```

Error handling

If it is possible for your User-Defined Function to produce an error, the Crystal Reports Formula Editor should be notified of the error. Many types of errors are automatically handled by the Formula Editor. Passing the wrong type of data to a function, for instance, will be recognized and trapped by Crystal Reports. However, if you design a function that can produce an error unique to that function, you should provide a means for reporting that error to the Formula Editor.

To send error messages to the Formula Editor, define the `UFErroText` string property in your Class Module. This property should be defined Public, using code similar to this:

```
Public UFErroText As String
```

Any time you trap for an error condition, simply set the `UFErroText` property to the error text you want reported in Crystal Reports. Setting the value of this property triggers the error in Crystal Reports, and Crystal Reports displays a dialog box containing the error message that you assigned to `UFErroText`.

Note: You should not use the `UFErroText` property for anything other than returning errors from User-Defined Functions. The `U2LCOM.DLL` UFL regularly resets the value of this property, so data can be lost if stored in `UFErroText` for any reason other than reporting an error.

Special purpose functions

You can define several special purpose functions in your Class Module in addition to the User-Defined Functions that you are creating. The Crystal Reports Formula Editor can look for and process code defined in any of the following functions:

```
Public Function UFIInitialize () As Long  
Public Function UFTerminate () As Long  
Public Sub UFStartJob (job As Long)  
Public Sub UFEndJob (job As Long)
```

Be sure to define the functions in your code exactly as they appear above. If not defined correctly, they will be ignored by Crystal Reports. These functions are completely optional when creating your Visual Basic Automation Server. They are provided to assist you with the design of your User-Defined Functions.

UFIInitialize

This function is called just after the DLL is loaded into memory. Use this function to handle one-time initialization of variables.

Return a value of 0 (zero) to indicate successful initialization. Any non-zero value indicates initialization failed.

UFTerminate

This function is called just before the DLL is unloaded from memory. Use this function to clean up any allocated memory or other data before unloading the DLL.

Return a value of 0 (zero) when finished cleaning up memory.

UFStartJob

This procedure is called by Crystal Reports just before User-Defined Functions are evaluated (or reevaluated). The Crystal Reports job number is passed to the function as the only parameter. Use this function to handle any initialization on a per-job basis.

UFEndJob

This procedure is called by Crystal Reports when the current job finishes, which happens when all pages of a report have been generated, before UFStartJob is called again, or before UFTerminate is called. This function also accepts the Crystal Reports job number as its only parameter. Handle any clean-up necessary on a per-job basis using this function.

Sample UFL Automation Server

Crystal Reports includes the source code for a sample automation server that exposes a User-Defined Function. The code includes a Visual Basic 4.0 project file that can be compiled in either Visual Basic 4.0 or 5.0. Use this sample as a reference for building your own Automation Server based User-Defined Functions. You can even copy the Class Module provided into your own projects as a head-start to building Automation Servers for U2LCOM.DLL.

Creating User-Defined Functions in Delphi 3.0

12

Crystal Reports allows you to create User-Defined Functions that are recognized by the Crystal Reports Formula Editor. In this chapter you will find detailed information on programming User-Defined Functions in Delphi.

Overview of User-Defined Functions in Delphi

The Crystal Reports Formula Editor and formula language are powerful tools, enabling you to do a wide variety of report-related tasks easily and efficiently. The formula language is expandable as well. That is, while it already includes a large selection of useful functions, it also comes with the ability to accept new functions that you define to meet your needs.

User-Defined Functions that are recognized by the Crystal Reports Formula Editor can be created in a Dynamic Link Library or, for 32-bit environments, in an Automation Server. This section demonstrates how to create User-Defined functions in an Automation Server using Borland Delphi 3.0 (and later versions). For information on how to create User-Defined Functions in a Dynamic Link Library using the C or C++ programming language, see [“Programming User-Defined Functions in C” on page 340](#). For information on how to create User-Defined Functions in an Automation Server using Visual Basic, see [“Creating User-Defined Functions in Visual Basic” on page 359](#).

Programming User-Defined Functions in Delphi

The Crystal Reports Formula Editor can access User-Defined Functions through a User Function Library (UFL). A User Function Library is a specially designed Dynamic Link Library that exposes one or more functions that you create. UFLs can be designed and programmed in any language that supports the development of Windows DLLs.

The 32-bit version of Crystal Reports Professional Edition includes the User Function Library U2LCOM.DLL. This UFL is installed in your `\Program Files\Common Files\crystal decisions\2.0\bin` directory when you install Crystal Reports and provides an interface through which you can expose User-Defined Functions in Automation Servers.

An Automation Server is a Dynamic Link Library or executable file that exposes its functionality to other modules and processes through the Component Object Model. For complete information on the Component Object Model (COM), refer to Microsoft documentation or the Microsoft World Wide Web site. U2LCOM.DLL is a UFL that can access any functions exposed by any Automation Servers named with a CRUFL prefix. This means that you can create an Automation Server in any language environment that supports COM, name the server with a CRUFL prefix, register the server on a system, and the Crystal Reports Formula Editor will access and make available any functions exposed by that Automation Server.

You may be more familiar with Automation Servers as OLE Automation. Though the technology behind Automation Servers is the same as OLE Automation, the correct name for these servers is now simply Automation Servers.

Delphi 3.0 allows you to design 32-bit Automation Servers easily. As a Delphi programmer, you can design User-Defined Functions for use in your reports by exposing them in an ActiveX Library and registering the library (Automation Server) on your system.

Using Delphi 3.0

Note: Version 3.0 of Delphi is required to create Automation Servers containing User-Defined Functions.

There are seven primary steps to creating User-Defined Functions in an automation server in Delphi 3.0:

- “Create the project” on page 373
- “Create the Automation Object” on page 373
- “Add Methods to the Type Library” on page 374
- “Register the Type Library” on page 374
- “Create the User-Defined Functions” on page 374
- “Build the project” on page 375
- “Using the User-Defined Functions” on page 375

Create the project

When Delphi first opens, it creates a default project and form for you.

- 1 Choose **New** from the **File** menu.
- 2 Click the **ActiveX Tab** in the New Items dialog box, and double-click the ActiveX Library icon.
Delphi creates a default ActiveX Library for you.
The U2LCOM.DLL UFL will only read functions exposed by Automation Servers named with a CRUFL prefix. For example, CRUFLMyFunctions is a valid project name for your Automation Server.
- 3 Choose **Save Project As** from the **File** menu, and save your Delphi project.
The project should be named CRUFLxxx.DPR, where xxx is a name of your choice.

Create the Automation Object

- 1 Choose **New** from the **File** menu.
- 2 Click the **ActiveX Tab** in the New Items dialog box, and double-click the Automation Object icon.
The Automation Object Wizard appears.
- 3 Enter a class name appropriate to the functions you will create.
- 4 Make sure Instancing is set to **Multiple Instance**, and click **OK**.
The Type Library Editor appears.

- 5 Make sure the name of the type library for your project matches the project name you created earlier. If not, change the name of the type library to CRUFLxxx, where xxx is the name you chose.

Add Methods to the Type Library

The methods in the class you specified when you created the type library will become User-Defined Functions that appear in the Crystal Reports Formula Editor.

- 1 Right-click the interface for your type library.
The interface name is identical to the class name you specified preceded by an I.
- 2 Choose **New** from the menu that appears, and then choose **Method**.
A new method appears below the interface in the Object list pane.
- 3 Name the method according to the function you want to create.
- 4 On the Attributes Tab for the method, declare your function.

For example:

```
function Square (Number: double): double;
```

Note: For complete information on how to declare functions in Delphi, refer to your Delphi documentation.

The U2LCOM.DLL UFL supports most standard Delphi data types and arrays. For complete information on how the Crystal Reports Formula Editor interprets Delphi data types and arrays, see [“Delphi and Crystal Reports” on page 376](#).

Continue creating methods for all User-Defined Functions you want to appear in the Crystal Reports Formula Editor.

Register the Type Library

- 1 Click **Register** in the Type Library Editor.
Delphi creates your type library and registers it on your system. A message should appear indicating that the ActiveX automation server was successfully registered.
- 2 Click **OK** in the message box.

Note: If the type library is not registered successfully on your system, create the type library and object methods over again. If you continue to have problems, refer to your Delphi documentation on creating type libraries.

Create the User-Defined Functions

- 1 Minimize the Type Library Editor, and locate the declaration of the method you created in the your type library in the Unit1.pas unit.
- 2 Enter code for the function as desired.

For example:

```
function TConversion.Square(Number: Double): Double;
begin
  result := Number * Number;
end;
```

- 3 Continue coding all methods that you declared for the interface.

Build the project

- Save all files in your project, and choose **Build All** from the **Project** menu.

Delphi builds your automation server, and the methods you declared are now available from the Crystal Reports Formula Editor.

Using the User-Defined Functions

- 1 In Crystal Reports, create a new report and add tables to it, or open an existing report.
- 2 From the **Insert** menu, choose **Formula Field**.
The Insert Field dialog box appears.
- 3 Click **New**, and enter a name for the new formula in the Formula Name dialog box.
- 4 Click **OK** and the Formula Editor appears.
- 5 Scroll down in the Functions list box to the Additional Functions section. Locate the User-Defined Function you created.
User-Defined Function names for functions created in Automation Servers are prefixed according to the project and class name used when you created the automation server. For instance, if you named your project *CRUFLProject*, named the class *Conversion*, and named the function *Square*, the User-Defined Function would appear in the Formula Editor as *ProjectConversionSquare*.
- 6 Double-click the **User-Defined Function**, and it appears in the Formula text box.
- 7 Enter valid arguments for the function.
For example:
`ProjectConversionSquare(5)`
- 8 Click **Check**.
Make sure no errors appear in the function.
- 9 Click **Accept**, and place the formula in your report.
- 10 Preview the report and verify that the function worked correctly.
Congratulations, you just created and used a User-Defined Function.

Delphi and Crystal Reports

Note: 32-bit Support Only. U2LCOM.DLL is a 32-bit UFL only and, therefore, supports only 32-bit Automation Server DLLs. To create User-Defined Functions for Crystal Reports, you must have Delphi 3.0 (or another 32-bit development environment that supports the creation of COM-based Automation Servers).

The following topics are discussed in this section:

- “Data types” on page 376.
- “Using arrays” on page 376.
- “Reserved names” on page 377.
- “Function name prefixing” on page 377.
- “Passing Parameters By Reference and By Value” on page 378.
- “Error handling” on page 378.
- “Special purpose functions” on page 378

Data types

Crystal Reports will support most common Delphi data types provided through a User-Defined Function developed in Delphi 3.0. The following table shows how Crystal Reports converts the most common Delphi data types to data types supported by the Formula Editor:

Delphi Data Types	Formula Editor Data Types
ShortInt Integer LongInt Real Single Double	NumberVar
Currency	CurrencyVar
Date	DateVar
Boolean	BooleanVar
String	StringVar

Note: Ranges—the range data type available in the Crystal Reports Formula Editor is not supported in COM-based User-Defined Functions.

Using arrays

Arrays can be passed to any User-Defined Function as a parameter of the function. This means that when you design your function in Delphi, the function can accept an array of values of any of the supported data types. However, the function

cannot return an array to the Crystal Reports Formula Editor. The following Delphi function is an acceptable User-Defined Function for Crystal Reports:

```
Function GetNthItem (A: MyArray, n: Integer): Integer;
begin
    GetNthItem := A[n];
End;
```

Reserved names

Certain names are reserved and cannot be used as User-Defined Function names. The following names are reserved by the Crystal Reports Formula Editor for special purposes:

- UFInitialize
- UFTerminate
- UFStartJob
- UFEndJob

For more explanation of these function names, see [“Special purpose functions” on page 378](#).

In addition, User-Defined Functions cannot use the same name as any of the functions exposed by the IDispatch interface used by COM:

- QueryInterface
- AddRef
- Release
- GetTypeInfoCount
- GetTypeInfo
- GetIDsOfNames
- Invoke

Function name prefixing

To ensure a unique name when User-Defined Functions appear in the Formula Editor, Crystal Reports appends a prefix to each function name that is generated from the project and class names. The first part of the prefix is the project name without the CRUFL prefix. The rest of the function name prefix is the class name.

Once the prefix for the function name is generated, all non-alphanumeric characters are removed, and the prefix is applied to the original function name. The following table illustrates this process:

Project Name	CRUFLTestFunctions
Class Name	Conversion
User-Defined Function Name	Date_To_String()
Formula Editor Function	TestFunctionsConversionDateToString()

Passing Parameters By Reference and By Value

Arguments can be passed to User-Defined Functions written in Delphi 3.0, either by value or by reference. For instance, both of the following functions are valid:

```
Function Test1 (num1: Integer): Integer;  
Function Test1 (var num1: Integer): Integer;
```

Error handling

If it is possible for your User-Defined Function to produce an error, the Crystal Reports Formula Editor should be notified of the error. Many types of errors are automatically handled by the Formula Editor. Passing the wrong type of data to a function, for instance, will be recognized and trapped by Crystal Reports. However, if you design a function that can produce an error unique to that function, you should provide a means for reporting that error to the Formula Editor.

To send error messages to the Formula Editor, define the `UFEErrorText` string property in your Class Module. This property should be defined Public, using code similar to this:

```
Property UFEErrorText: String;
```

Any time you trap for an error condition, simply set the `UFEErrorText` property to the error text you want reported in Crystal Reports. Setting the value of this property triggers the error in Crystal Reports, and Crystal Reports displays a dialog box containing the error message that you assigned to `UFEErrorText`.

Note: You should not use the `UFEErrorText` property for anything other than returning errors from User-Defined Functions. The `U2LCOM.DLL` UFL regularly resets the value of this property, so data can be lost if stored in `UFEErrorText` for any reason other than reporting an error.

Special purpose functions

You can define several special purpose functions in your Class Module in addition to the User-Defined Functions that you are creating. The Crystal Reports Formula Editor can look for and process any of the following class methods:

```
Function UFInitialize: Integer;  
Function UFTerminate: Integer;  
Procedure UFStartJob (job: Integer)  
Procedure UFEndJob (job: Integer)
```

Be sure to declare the methods in your code exactly as they appear above. If not declared correctly, they will be ignored by Crystal Reports. These methods are completely optional when creating your Delphi Automation Server. They are provided to assist you with the design of your User-Defined Functions.

UFInitialize

This function is called just after the DLL is loaded into memory. Use this function to handle one-time initialization of variables.

Returns a value of 0 (zero) to indicate successful initialization. Any non-zero value indicates initialization failed.

UF5Initialize

This function is similar to UFInitialize except that it takes a version number.

UFTerminate

This function is called just before the DLL is unloaded from memory. Use this function to clean up any allocated memory or other data before unloading the DLL.

Returns a value of 0 (zero) when finished cleaning up memory.

UFStartJob

This procedure is called by Crystal Reports just before User-Defined Functions are evaluated (or reevaluated). The Crystal Reports job number is passed to the function as the only parameter. Use this function to handle any initialization on a per-job basis.

UFEndJob

This procedure is called by Crystal Reports when the current job finishes, which happens when all pages of a report have been generated, before UFStartJob is called again, or before UFTerminate is called. This function also accepts the Crystal Reports job number as its only parameter. Handle any clean-up necessary on a per-job basis using this function.

Deprecated and Retired API Reference 13

The following developer APIs, interfaces, methods, and properties are retired or deprecated. Retired features are no longer available or supported. Deprecated features, while they may be supported for backwards compatibility, are not recommended for use in the current version of Crystal Reports. Deprecated features may become obsolete in future versions of the product.

Retired Developer APIs

Crystal Reports provides many new options to meet your development needs as you move from single-tier to multi-tier to Enterprise applications. The *Developer's Guide*, provided with the developer editions of Crystal Reports, outlines these new options. It also provides an overview of the RDC, which continues to be the standard for desktop applications.

Some of the older Developer APIs, however, have been retired as shown in the following list.

Crystal Reports ActiveX Control (crystl32.ocx)

The Crystal Reports ActiveX control is no longer supported and no longer available as of Crystal Reports version 9.

For information on migrating from the Crystal Reports ActiveX control to the RDC, go to <http://support.crystaldecisions.com/library/> and enter the file name scr8-ocxtordc.pdf.

Crystal Reports Automation Server (cpeaut32.dll)

The Crystal Reports Automation Server is no longer supported and no longer available as of Crystal Reports version 9.

For information on migrating from the Crystal Reports Automation Server to the RDC, go to <http://support.crystaldecisions.com/library/> and enter the file name scr8-cpeauttordc.pdf.

Crystal Reports Print Engine (crpe32.dll)

The Crystal Reports Print Engine is now considered a legacy API and no longer exposes calls for any of the new features included in Crystal Reports. The API is still available to developers. Help is located in *Crystal Reports Legacy SDK Help (Legacy.chm)*.

RDC Runtime C Headers

The RDC runtime C headers are no longer supported as of Crystal Reports version 9.

Design Time Control for Microsoft Visual Interdev (DTC)

The DTC is no longer supported as of Crystal Reports version 9.

Retired P2smon.dll functions

The Active Data driver (p2smon.dll) is no longer supported as of Crystal Reports version 9. Active data is now handled through individual database drivers. For more information see “Active Data” in the *Crystal Reports Developer Runtime Help (Runtime.hlp)*.

In order to retain backwards compatibility of its runtime functions, the p2smon.dll is available in the \tools directory on the Crystal Reports installation CD. Include the p2smon.dll if you have any the following functions in your application:

- “CreateFieldDefFile” on page 383
- “CreateReportOnRuntimeDS” on page 384
- “SetActiveDataSource” on page 385

CreateFieldDefFile

This is a retired function. It creates a tab-separated text file, known as a field definition file, which represents the structure of the data in a specified Recordset or Rowset object. This field definition file can then be used to design a report file using the Runtime DB option in the Create Report Expert of Crystal Reports. When designing an application that prints, previews, or exports the report, the field definition file can be replaced, at runtime, by the Recordset or Rowset object.

C Syntax

```
BOOL FAR PASCAL CreateFieldDefFile(LPUNKNOWN FAR *lpUnk,
    LPCSTR fileName,
    BOOL bOverWriteExistingFile);
```

Visual Basic Syntax

```
Declare Function CreateFieldDefFile Lib "p2smon.dll" (lpUnk As Object, _
    ByVal fileName As String, ByVal bOverWriteExistingFile As Long) _
    As Long
```

Parameters

Parameter	Description
lpUnk	The active data source used to create the field definition file. In C or C++, this is a pointer to an unknown derived COM interface relating to a DAO or ADO Recordset. In Visual Basic, this is a Recordset or Rowset object.
fileName	The path and file name of the field definition file to be created.
bOverWriteExistingFile	If a field definition file already exists with the specified path and file name, this flag indicates whether or not to overwrite that file.

Return Value

Returns 0 (False) if the call failed. Returns 1 (True) if the call succeeded and the field definition file was created.

Remarks

This function creates a field definition file only, and does not create a report file. You must create a report file using Crystal Reports.

CreateReportOnRuntimeDS

This is a retired function. It creates a tab-separated text file, known as a field definition file, which represents the structure of the data in a specified Recordset or Rowset object. Then, the function creates a blank report file based on this field definition file. When designing an application that prints, previews, or exports the report, the field definition file can be replaced by the Recordset or Rowset in the active data source.

C Syntax

```
BOOL FAR PASCAL CreateReportOnRuntimeDS(LPUNKNOWN FAR *lpUnk,
    LPCSTR reportFile,
    LPCSTR fieldDefFile,
    BOOL bOverWriteFile,
    BOOL bLaunchDesigner);
```

Visual Basic Syntax

```
Declare Function CreateReportOnRuntimeDS Lib "p2smon.dll" ( _
    lpUnk As Object, ByVal reportFile As String, ByVal fieldDefFile _
    As String, ByVal bOverWriteFile As Long, ByVal bLaunchDesigner _
    As Long) As Long
```

Parameters

Parameter	Description
lpUnk	The active data source used to create the field definition file. In C or C++, this is a pointer to an Unknown derived COM interface relating to a DAO or ADO Recordset. In Visual Basic, this is a Recordset or Rowset object.
reportFile	The path and file name of the report file to be created.
fieldDefFile	The path and file name of the field definition file to be created.
bOverWriteFile	If a field definition file already exists with the specified path and file name, this flag indicates whether or not to overwrite that file.
bLaunchDesigner	If True (1), Crystal Reports is launched with the newly created report file opened. Crystal Reports must be installed on the system.

Return Value

Returns 0 (False) if the call failed. Returns 1 (True) if the call succeeded and the field definition file was created.

Remarks

This function creates a field definition file, then creates a report file based on that field definition file. The function `CreateFieldDefFile` is unnecessary when this function is used.

SetActiveDataSource

This is a retired function. It is used to provide information about a data source to the Crystal Active Data database driver. For instance, if a report has been designed using the Crystal Active Data Driver, this method can be used to provide an active data source for the report, such as a DAO, ADO, or RDO recordset or a CDO rowset. In this case, the object passed to the third parameter of this function replaces, at runtime, the field definition file used to create the report.

Visual Basic syntax

```
Declare Function SetActiveDataSource Lib "p2smon.dll" _
    (ByVal printJob as Integer, ByVal tableNum as Integer, x as Object) _
    As Long
```

Parameters

Parameter	Description
printJob	Specifies the print job to which you want to add the active data source.
tableNum	The 0 based number of a table for which you want to pass the active data recordset or rowset.
x	Variant data passed to the database driver such as a DAO, ADO or RDO recordset or a CDO rowset.

Return Value

Returns 0 (False) if the call failed. Returns 1 (True) if the call succeeded the data source was passed.

Remarks

The `SetActiveDataSource` function is used in conjunction with the Crystal Report Print Engine (`Crpe32.dll`) in a Visual Basic application. If you are using Visual C++, see “`PESetNthTablePrivateInfo`” in the *Crystal Reports Legacy SDK Help (Legacy.chm)*.

Deprecated RDC Interfaces

Deprecated interfaces	Explanation
IFieldDefinition Interface	This interface was originally defined to provide generic properties for the various types of field definition objects. It is still the base for all field definition objects. In lieu of this object, use the “FieldDefinitions Collection” on page 61 . After using the Item property to get a field definition object from the Collection, you can use the Kind property to determine what type of field definition object the item is. All field definition objects have the Kind property.
IReportObject	This interface was originally defined to provide generic properties for the various types of report objects. It is still the base for all report objects. In lieu of this object, use the “ReportObjects Collection” on page 133 . After using the Item property to get a report object from the Collection, you can use the Kind property to determine what type of report object the item is. All report objects have the Kind property.

IFieldDefinition Interface

This is a deprecated interface. It was originally defined to provide generic properties for the various types of field definition objects. It is still the base for all field definition objects. In lieu of this object, use the [“FieldDefinitions Collection” on page 61](#). After using the Item property to get a field definition object from the Collection, you can use the Kind property to determine what type of field definition object the item is. All field definition objects have the Kind property.

IFieldDefinition Object Properties

Property	Description	Read/Write	Restriction in event handler
Kind	“CRFieldKind” . Gets field definition kind.	Read only	None
Name	String. Gets field definition unique formula name.	Read only	None
NextValue	Variant. Gets the field next value.	Read only	None
NumberOfBytes	Integer. Gets field number of bytes.	Read only	None
Parent	“Report Object” . Gets reference to the parent object.	Read only	None
PreviousValue	Variant. Gets the field previous value.	Read only	None

Property	Description	Read/Write	Restriction in event handler
UseCount	Long. Gets the field use count.	Read only	None
Value	Variant. Gets the field current value.	Read only	None
ValueType	"CRFieldValueType". Gets the field value type.	Read only	None

IReportObject

This is a deprecated interface. It was originally defined to provide generic properties for the various types of report objects. It is still the base for all report objects. In lieu of this object, use the ["ReportObjects Collection" on page 133](#). After using the Item property to get a report object from the Collection, you can use the Kind property to determine what type of report object the item is. All report objects have the Kind property

IReportObject Properties

Property	Description	Read/Write	Restriction in event handler
BackColor	OLE_COLOR. Gets or sets the object background color.	Read/Write	Can be written only when formatting idle or active.
BorderColor	OLE_COLOR. Gets or sets the object border color.	Read/Write	Can be written only when formatting idle or active.
BottomLineStyle	"CRLLineStyle". Gets or sets bottom line style.	Read/Write	Can be written only when formatting idle or active.
CloseAtPageBreak	Boolean. Gets or sets the close border on page break option.	Read/Write	Can be written only when formatting idle or active.
EnableTightHorizontal	Boolean. Gets or sets the enable tight horizontal option.	Read/Write	Can be written only when formatting idle or active.
HasDropShadow	Boolean. Gets or sets the border drop shadow option.	Read/Write	Can be written only when formatting idle or active.
Height	Long. Gets or sets object height, in twips.	Read/Write	Can be written only when formatting idle or active.
KeepTogether	Boolean. Gets or sets the keep object together option.	Read/Write	Can be written only when formatting idle or active.
Kind	"CROBJECTKind". Gets which kind of object (for example, box, cross-tab, field).	Read only	None
Left	Long. Gets or sets the object upper left position, in twips.	Read/Write	Can be written only when formatting idle or active.
LeftLineStyle	"CRLLineStyle". Gets or sets the left line style.	Read/Write	Can be written only when formatting idle or active.

Property	Description	Read/Write	Restriction in event handler
Name	String. Gets or sets the object name.	Read/Write	Can be written only when formatting idle or active.
NeedUpdated Pages	Gets whether the user needs to update pages for display due to changes in the report.	Read only	None
Parent	"Section Object". Reference to the parent object.	Read only	Can be written only when formatting idle or active.
RightLineStyle	"CRLLineStyle". Gets or sets the right line style.	Read/Write	Can be written only when formatting idle or active.
Suppress	Boolean. Gets or sets the object visibility.	Read/Write	Can be written only when formatting idle or active.
Top	Long. Gets or sets the object upper top position, in twips.	Read/Write	Can be written only when formatting idle or active.
TopLineStyle	"CRLLineStyle". Gets or sets the top line style.	Read/Write	Can be written only when formatting idle or active.
Width	Long. Gets or sets the object width, in twips.	Read/Write	Can be written only when formatting idle or active.

Deprecated RDC Properties

Deprecated properties	Explanation
IndexUsed	Deprecated as of Crystal Reports 9, with no replacement. Integer. This property was originally defined in the TableLink Object to get the table link index used.
LogonDatabaseName	This property was originally defined in the DatabaseTable Object to get the logon database name. In lieu of this method, use the name value pair in the ConnectionProperty Object . For example: Name is "Database Name" and the value is the file path and name of the data source. Note: The name value pair will depend on the data source accessed.
LogonServerName	This property was originally defined in the DatabaseTable Object to get the logon server name. In lieu of this method, use the name value pair in the ConnectionProperty Object . For example: Name is "Server" and the value is the server name of the data source. Note: The name value pair will depend on the data source accessed.

Deprecated properties	Explanation
LogonUserID	<p>This property was originally defined in the DatabaseTable Object to get the logon user ID. In lieu of this method, use the name value pair in the ConnectionProperty Object. For example: Name is "User ID" and the value is the user name required to connect to the data source.</p> <p>Note: The name value pair will depend on the data source accessed.</p>
MorePrintEngineError Messages	<p>Deprecated as of Crystal Reports 9, with no replacement. Boolean. This property was originally defined in the Report Object to get or set the report option that indicates whether to pop up database error dialogs during printing when a Report Engine error occurs.</p>
PartialMatchEnabled	<p>Deprecated as of Crystal Reports 9, with no replacement. Boolean. This property was originally defined in the TableLink Object to get the table link partial-match enabled option.</p>
SessionUserID	<p>This property was originally defined in the DatabaseTable Object to get the session user ID. In lieu of this method, use the name value pair in the ConnectionProperty Object where name is "Session UserID" and the value is the user name required to connect to the data source.</p>
SubLocation	<p>This property was originally defined in the DatabaseTable Object to get the table sublocation. In lieu of this method, use the name value pair in the ConnectionProperty Object where name is "Table Name" and value is the name of the database table.</p>
TranslateDosMemos	<p>Deprecated as of Crystal Reports 9, with no replacement. Boolean. This property was originally defined in the Report Object to gets or set the report option that indicates whether to translate DOS memos.</p>
TranslateDosStrings	<p>Deprecated as of Crystal Reports 9, with no replacement. Boolean. This property was originally defined in the Report Object to get or set the report option that indicates whether to translate DOS strings.</p>

Deprecated RDC Methods

Deprecated methods	Explanation
ConvertDatabaseDriver Method (Database Object)	This is a deprecated method. This method was originally defined to convert the database driver DLL used by the report. In lieu of this method, use the DLLName property of the DatabaseTable Object and the Add method, Delete method and DeleteAll method of the ConnectionProperties Collection to reset the connection properties.
SetLogOnInfo Method (DatabaseTable Object)	This method was originally defined to log on to the data source so table data can be accessed. In lieu of this method, use the name value pairs in the ConnectionProperty Object to provide the connection information needed to log on to the data source. Note: The name and value pairs will depend on the data source accessed.
SetMorePrintEngineErrorMessages Method (Application Object)	Deprecated as of Crystal Reports 9, with no replacement. This method was originally defined to set the global more print engine error messages option.
SetSessionInfo Method (DatabaseTable Object)	This method was originally defined to allow the user to log on to a secured Access session. In lieu of this method, use the name value pairs in the ConnectionProperty Object . These would be "Session UserID" where the value is the session's user name and "Session Password" where the value is the session's password.

ConvertDatabaseDriver Method (Database Object)

This is a deprecated method. This method was originally defined to convert the database driver DLL used by the report. In lieu of this method, use the DLLName property of the "[DatabaseTable Object](#)" on [page 50](#) and the Add method, Delete method and DeleteAll method of the "[ConnectionProperties Collection](#)" on [page 35](#) to reset the connection properties.

Syntax

```
Sub ConvertDatabaseDriver (pDLLName As String, bDoImmediateConvert as Boolean)
```


Parameters

Parameter	Description
pDllName	Specifies the new DLL for the database driver.
blDoImmediateConvert	Specifies when the database driver will be converted. If True, the database driver is converted when the report is previewed, or the when the database is verified. If False, the database driver is converted when the report is refreshed in the preview window.

Remarks

- Set blDoImmediate to False when you want the user to be prompted for the data source information. This applies to reports that are previewed and refreshed in the Crystal Report Viewer, and to reports that are exported or saved to a Crystal Report format (.rpt), and previewed and refreshed in the Crystal Reports Designer. When the report is previewed, a dialog box for the new data source appears. For example, converting to ODBC (P2sodbc.dll) opens the Set Data Source dialog box, while converting to OLEDB (P2soledb.dll) opens the Data Link Properties dialog box.
- Set blDoImmediateConvert to True when you want to set the data source at runtime. Additional code must be added to set the logon information for the data source, and to verify the database. One exception is converting to OLEDB (P2soledb.dll). Since there is no method to set the provider at runtime the Data Links dialog box will always prompt for the data source. In this case only the code to convert the database driver is required.
- Once the database is converted, and the data source is set, either through code, or through a dialog box, three additional prompts may appear:
 - Logon to the data source
This dialog box allows you to log on to the correct data source. If the data source is not secured, you can click OK without entering any logon information.
 - Verify the database
This is a message warning the user that the database file has changed and the report is being updated.
 - Map the fields
This dialog box allows you to map the fields that have changed with the new data source. For more information on mapping fields see "Remapping processes" in the *Crystal Reports Online Help (crw.chm)*.

Sample

The following code demonstrates how to convert the database driver to ODBC (P2sodbc.dll) using the ConverDataBaseDriver method. The ODBC data source is pointing to a Microsoft Access database.

```
'Instantiate the report object.
Set m_Report = New CrystalReport1

' Convert the database driver to ODBC.
m_Report.Database.ConvertDatabaseDriver "p2sodbc.dll", True

' Set the logon information for the ODBC data source.
' If the logon information is not set, an error will be produced when the
' report is previewed or exported.
m_Report.Database.Tables(1).SetLogOnInfo "Xtreme Sample Database", "",
"", ""

' Verify the database.
' If the database is not verified before exporting, an error will be
produced.
' If the database is not verified before previewing the report, the user
may be
' prompted when the report is refreshed in the Crystal Report Viewer.
m_Report.Database.Verify
```

SetLogOnInfo Method (DatabaseTable Object)

This is a deprecated method. This method was originally defined to log on to the data source so table data can be accessed. In lieu of this method, use the name value pairs in the [“ConnectionProperty Object”](#) on page 29 to provide the connection information needed to log on to the data source.

Note: The name and value pairs will depend on the data source accessed.

Syntax

```
Sub SetLogOnInfo (pServerName As String,
[pDatabaseName], [pUserID], [pPassword])
```

Parameters

Parameter	Description
pServerName	Specifies the name of the server or ODBC data source where the database is located (that is, CRSS).
[pDatabaseName]	Specifies the name of the database.
[pUserID]	Specifies a valid user name for logging on to the data source.
[pPassword]	Specifies a valid password for logging on to the data source.

SetMorePrintEngineErrorMessages Method (Application Object)

This is a deprecated method as of Crystal Reports 9, with no replacement. This method was originally defined to set the global more print engine error messages option.

Syntax

```
Sub SetMorePrintEngineErrorMessages (bl As Boolean)
```

Parameter

Parameter	Description
bl	Specifies whether the option is selected (TRUE)

SetSessionInfo Method (DatabaseTable Object)

This is a deprecated method. This method was originally defined to allow the user to log on to a secured Access session. In lieu of this method, use the name value pairs in the [“ConnectionProperty Object” on page 29](#). These would be “Session UserID” where the value is the session’s user name and “Session Password” where the value is the session’s password.

Syntax

```
Sub SetSessionInfo (pSessionUserID As String, pSessionPassword As String)
```

Parameters

Parameter	Description
pSessionUserID	Specifies the Access userID used to log on to an Access session.
pSessionPassword	Specifies the session password for Access secured session.

Remarks

In Microsoft Access 95 and later, an Access database can have session security (also known as user-level security), database-level security, or both. If the Access database contains only session security, simply pass the session password to the SessionPassword parameter. If the Access database contains database-level security, use a linefeed character, Chr(10), followed by the database-level password. For example:

```
object.SetSessionInfo “userID”, Chr(10) & “dbpassword”
```

If the Access database contains both session security and database-level security, use the session password followed by the linefeed character and the database password.

```
object.SetSessionInfo “userID”, _  
    “sesspswd” & Chr(10) & “dbpassword”
```

Alternately, database-level security can also be handled by assigning the database-level password to the Password parameter of the [SetLogOnInfo Method \(DatabaseTable Object\)](#).

Deprecated RDC Export Format Types

Deprecated format type	Explanation
Data Interchange Format	Deprecated as of Crystal Reports 9, with no replacement. This export format type is now obsolete and was set through the FormatType property of the ExportOptions Object using the crEFTDataInterchange constant. The export format DLL u2fdif.dll is no longer shipped with Crystal Reports.
Lotus Works format	Deprecated as of Crystal Reports 9, with no replacement. This export format type is now obsolete and was set through the FormatType property of the ExportOptions Object using the crEFTLotus123WK1, crEFTLotus123WK3, and crEFTLotus123WKS constants. The export format DLL u2works.dll is no longer shipped with Crystal Reports.

Index

A

ActivateView method	247
Active Data Driver	
using	299
working with	298
ActiveX	
Crystal Data Object	308
Crystal Report Viewers	233
Report Viewer Object Model	244
ActiveX Data Sources	298
using at design time	305
Add method	
ConnectionProperties Collection	35
CrossTabGroups Collection	39
DatabaseTables Collection	54
FieldDefinitions Collection	62
FieldElements Collection	67
FormulaFieldDefinitions Collection	76
ObjectSummaryFieldDefinitions Collection	88
ParameterValueInfos Collection	114
ParameterValues Collection	113
ReportAlerts Collection	130
RunningTotalFieldDefinitions Collection	138
Sections Collection	150
SortFields Collection	152
SQLExpressionFieldDefinitions Collection	156
SubreportLinks Collection	158
SummaryFieldDefinitions Collection	164
TableLinks Collection	166
AddADOCCommand method	43
AddBlobFieldObject method	141
AddBoxObject method	141
AddCrossTabObject method	142
AddCurrentRange method	107
AddCurrentValue method	108
AddDefaultValue method	108
AddField method	329
AddFieldObject method	142
AddGraphObject method	143
AddGroup method	120
AddLineObject method	143
AddOLEDBSource method	43
AddParameter method	264
AddParameterEx method	264
AddPictureObject method	144
AddReportVariable method	121
AddRows method	330

AddSpecialVarFieldObject method	144
AddStoredProcedure method	55
AddSubreportObject method	145
AddSummaryFieldObject method	145
AddTextObject method	146
AddUnboundFieldObject method	147
AddView method	247
ADO data sources	298
AfterFormatPage Event	127
applets, Crystal Report Viewer	241
application object, sample Code in VB	17
Automation Server	
registration and distribution in a VB project	364
sample Code in VB	369
AutoSetUnboundFieldSource method	121

B

BeforeFormatPageEvent	128
button conventions	6

C

CancelPrinting method	122
CanClose method	17
Cascading Style Sheets	9
CDO, see Crystal Data Object	
Check method	
FormulaFieldDefinition Object	75
SQLExpressionFieldDefinition Object	155
CheckDifferences method	51
ClearCurrentValueAndRange method	109
Clicked Event	253
CloseButtonClicked Event	253
CloseView method	248
Collections	
Areas	25
ConnectionProperties	35
CRFields	244
CrossTabGroups	38
DatabaseFieldDefinitions	49
DatabaseTables	53
FieldDefinitions	61
FieldElements	66
FormulaFieldDefinitions	75
GroupNameFieldDefinitions	84
ObjectSummaryFieldDefinitions	87
Pages	103
ParameterFieldDefinitions	111

ParameterValueInfos.....	114	CRFieldKind Enum.....	208
ParameterValues.....	112	CRFieldMappingType Enum	208
Report Designer.....	16	CRFieldType Enum	265
ReportObjects	133	CRFieldValueType Enum	209
RunningTotalFieldDefinitions	137	CRFontColorConditionFormulaType Enum	209
Sections.....	150	CRFormulaSyntax Enum	210
SortFields.....	152	CRGraphColor Enum	210
SQLExpressionFieldDefinitions	155	CRGraphDataPoint Enum	210
SubreportLinks.....	158	CRGraphDataType Enum.....	210
SummaryFieldDefinitions	164	CRGraphDirection Enum	210
TableLinks	166	CRGraphType Enum	211
COM data provider.....	320	CRGridlineType Enum.....	212
adding		CRGroupCondition Enum	212
a class module.....	322	CRHourType Enum.....	213
a reference to MS ADO Library.....	323	CRHTMLPageStyle Enum	213
a Sub Main() procedure	323	CRHTMLToolbarStyle Enum	214
advantages	321	CRImageType Enum	214
compiling the ActiveX DLL.....	324	CRLeadingDayPosition Enum	214
creating		CRLeadingDayType Enum	214
a new project.....	322	CRLegendPosition Enum.....	214
a report from.....	325	CRLineSpacingType Enum	215
functions.....	323	CRLineStyle Enum	215
command conventions.....	6	CRLinkJoinType Enum	215
CRAlignment Enum.....	201	CRLinkLookUpType Enum.....	215
CRAMPMType Enum	201	CRLoadingType Enum	265
CRAreaKind Enum	202	CRMarkerShape Enum.....	216
CRBarSize Enum	202	CRMarkerSize Enum.....	216
CRBindingMatchType Enum	202	CRMinuteType Enum.....	216
CRBooleanFieldFormatConditionFormulaType		CRMonthType Enum.....	216
Enum	202	CRNegativeType Enum	217
CRBooleanOutputType Enum	203	CRNumberFormat Enum.....	217
CRBorderConditionFormulaType Enum	203	CRNumericFieldFormatConditionFormulaType	
CRCommonFieldFormatConditionFormulaType		Enum	217
Enum	203	CRObjectFormatConditionFormulaType Enum	218
CRConvertDateTimeType Enum.....	203	CRObjectKind Enum.....	218
CRCurrencyPositionType Enum	204	CRObjectType Enum	266
CRCurrencySymbolType Enum	204	CROpenReportMethod Enum	219
CRDatabaseType Enum.....	204	CRPaperOrientation Enum.....	219
CRDataSource	335	CRPaperSize Enum	219
CRDateCalendarType Enum.....	204	CRPaperSource Enum	220
CRDateEraType Enum.....	205	CRParameterFieldType Enum.....	221
CRDateFieldFormatConditionFormulaType Enum	205	CRParameterPickListSortMethod Enum	221
CRDateOrder Enum	205	CRPieLegendLayout Enum	221
CRDateTimeFieldFormatConditionFormulaType		CRPieSize Enum	222
Enum	206	CRPlaceholderType Enum.....	222
CRDateWindowsDefaultType Enum	206	CRPrinterDuplexType Enum	222
CRDayType Enum.....	206	CRPrintingProgress Enum.....	222
CRDiscreteOrRangeKind Enum	206	CRRangeInfo Enum.....	223
CRDivisionMethod Enum.....	206	CRRenderResultType Enum	223
CRDrillType Enum	265	CRReportFileFormat Enum.....	223
Create Report Expert, database definition tool	304	CRReportKind Enum	223
CreatePageGenerator method	95	CRReportVariableValueType Enum.....	223
CreateSubreportPageGenerator method	99	CRRotationAngle Enum	224
CRExchangeDestinationType Enum	207	CRRoundingType Enum.....	224
CRExportDestinationType Enum	207	CRRunningTotalCondition Enum	225
CRExportFormatType Enum	207	CRSearchDirection Enum	225

CRSecondType Enum	225
CRSectionAreaFormatConditionFormulaType Enum	225
CRSliceDetachment Enum	226
CRSortDirection Enum	226
CRSpecialVarType Enum	226
CRStringFieldConditionFormulaType Enum	227
CRSubreportConditionFormulaType Enum	227
CRSummaryType Enum	227
CRTableDifferences Enum	228
CRTextFormat Enum	228
CRTimeBase Enum	229
CRTimeFieldFormatConditionFormulaType Enum	229
CRTopOrBottomNGroupSortOrder Enum	229
CRTrackCursor Enum	267
CRValueFormatType Enum	229
CRViewer Object	256
CRViewer object	234
CRViewingAngle Enum	230
CRYearType Enum	230
Crystal Data Object	308
using	309
Crystal Data Object Model	311
Crystal Data Objects	328
Crystal Data Source Type Library	311, 335
Crystal Data Sources	320
Crystal DataSouce object, passing to Active Data Driver	318
Crystal Report Viewer, See Report Viewer	
CrystalComObject	328

D

DAO data sources	298
Data Definition Files	302
creating	303
data security	235
database definition tool	304
DbClicked Event	253
Delete method	
ConnectionProperties Collection	36
CrossTabGroups Collection	39
DatabaseTables Collection	56
FieldDefinitions Collection	62
FieldElements Collection	67
FormulaFieldDefinitions Collection	76
ObjectSummaryFieldDefinitions Collection	88
ParameterValueInfos Collection	115
ParameterValues Collection	113
ReportAlerts Collection	131
RunningTotalFieldDefinitions Collection	138
Sections Collection	151
SortFields Collection	153
SQLExpressionFieldDefinitions Collection	156
SubreportLinks Collection	159
SummaryFieldDefinitions Collection	165
TableLinks Collection	167

DeleteAll method	37
DeleteField method	331
DeleteGroup method	122
DeleteNthDefaultValue method	109
DeleteObject method	147
Delphi, see User-Defined Functions deprecated	
Developer APIs	382
P2smon.dll functions	383
RDC Export Format Types	394
RDC Interfaces	386
RDC Methods	390
RDC Properties	388
DiscardSavedData method	122
DownloadFinished Event	254
DownloadStarted Event	254
DrillOnDetail Event	255
DrillOnGraph Event	255
DrillOnGraph method	99
DrillOnGroup Event	255
DrillOnMap method	100
DrillOnSubreport Event	256
DrillOnSubreport method	100
Drivers	
database (ADO, DAO, and RDO)	298
using Active Data	299

E

Embeddable Designer	292
designing a report with	288
developing applications with	292
distributing	293
properties and methods	294
providing online help to the user	293
sample application	
Visual Basic	274
Visual C++	280
enhancements, Report Designer Component v. 9	7
Enumerated Types (Report Designer)	
CRAlignment	201
CRAMPMTType	201
CRAreaKind	202
CRBarSize	202
CRBindingMatchType	202
CRBooleanFieldFormatConditionFormulaType ..	202
CRBooleanOutputType	203
CRBorderConditionFormulaType	203
CRCommonFieldFormatConditionFormulaType ..	203
CRConvertDateTimeType	203
CRCurrencyPositionType	204
CRCurrencySymbolType	204
CRDatabaseType	204
CRDateCalendarType	204
CRDateEraType	205
CRDateFieldFormatConditionFormulaType	205

Enumerated Types (Report Designer) *continued*

CRDateOrder	205
CRDateTimeFieldFormatConditionFormula Type	206
CRDateWindowsDefaultType	206
CRDayType	206
CRDiscreteOrRangeKind	206
CRDivisionMethod	206
CRExchangeDestinationType	207
CRExportDestinationType	207
CRExportFormatType	207
CRFieldKind	208
CRFieldMappingType	208
CRFieldValueType	209
CRFontColorConditionFormulaType	209
CRFormulaSyntax	210
CRGraphColor	210
CRGraphDataPoint	210
CRGraphDataType	210
CRGraphDirection	210
CRGraphType	211
CRGridlineType	212
CRGroupCondition	212
CRHourType	213
CRHTMLPageStyle	213
CRHTMLToolbarStyle	214
CRImageType	214
CRLeadingDayPosition	214
CRLeadingDayType	214
CRLegendPosition	214
CRLineSpacingType	215
CRLineStyle	215
CRLinkJoinType	215
CRLinkLookUpType	215
CRMarkerShape	216
CRMarkerSize	216
CRMinuteType	216
CRMonthType	216
CRNegativeType	217
CRNumberFormat	217
CRNumericFieldFormatConditionFormula Type	217
CRObjectFormatConditionFormulaType	218
CROBJECTKind	218
CROpenReportMethod	219
CRPaperOrientation	219
CRPaperSize	219
CRPaperSource	220
CRParameterFieldType	221
CRParameterPickListSortMethod	221
CRPieLegendLayout	221
CRPieSize	222
CRPlaceholderType	222
CRPrinterDuplexType	222
CRPrintingProgress	222
CRRangeInfo	223

CRRenderResultType	223
CRReportFileFormat	223
CRReportKind	223
CRReportVariableValueType	223
CRRotationAngle	224
CRRoundingType	224
CRRunningTotalCondition	225
CRSearchDirection	225
CRSecondType	225
CRSectionAreaFormatConditionFormulaType	225
CRSliceDetachment	226
CRSortDirection	226
CRSpecialVarType	226
CRStringFieldConditionFormulaType	227
CRSubreportConditionFormulaType	227
CRSummaryType	227
CRTableDifferences	228
CRTextFieldFormat	228
CRTIMEBase	229
CRTIMEFieldFormatConditionFormulaType	229
CRTopOrBottomNGroupSortOrder	229
CRValueFormatType	229
CRViewingAngle	230
CRYEARType	230
Enumerated Types (Report Viewer)	
CRDrillType	265
CRFieldType	265
CRLoadingType	265
CROBJECTType	266
CRTrackCursor	267
error table	349
errors	
User-Defined Functions in C.	351
User-Defined Functions in Delphi	378
User-Defined Functions in VB	367
Events (JavaBean)	272
ServerRequestEvent,	272
ViewChangeEvent	272
Events (Report Designer)	
AfterFormatPage	127
BeforeFormatPage	128
FieldMapping	128
Format	148
NoData	128
Report Object	127
Section Object	148
Events (Report Viewer)	236
Clicked	253
CloseButtonClicked	253
CRViewer Object	252
DblClicked	253
DownloadFinished	254
DownloadStarted	254
DrillOnDetail	255
DrillOnGraph	255
DrillOnGroup	255

DrillOnSubreport	256	GetLicenseStatus method	18
ExportButtonClicked	256	GetNextRows method	123
FirstPageButtonClicked	257	GetNthCurrentRange method	109
GoToPageNButtonClicked	257	GetNthCurrentValue method	110
GroupTreeButtonClicked	257	GetNthDefaultValue method	110
HelpButtonClicked	258	GetPageNumberForGroup method	101
LastPageButtonClicked	258	GetReportVariableValue method	123
NextPageButtonClicked	258	GetVersion method	18
OnReportSourceError	258	GetViewName method	248
PrevPageButtonClicked	259	GetViewPath method	249
PrintButtonClicked	259	GoToPageNButtonClicked Event	257
RefreshButtonClicked	259	GroupTreeButtonClicked Event	257
SearchButtonClicked	260		
SearchExpertButtonClicked	260	H	
SelectionFormulaBuilt	260	help	
SelectionFormulaButtonClicked	261	product registration	5
ShowGroup	261	technical support	5
StopButtonClicked	261	HelpButtonClicked Event	258
ViewChanged	262	Helper Modules	344
ViewChanging	262		
ZoomLevelChanged	263	I	
Export method		ImportSubreport method	147
Report Designer PageGenerator Object	101	InitForJob	350
Report Designer Report Object	122	implementing	357
ExportButtonClicked Event	256	see also User-Defined Functions, programming in C	
F		J	
fax numbers, registration	5	Java	
FieldMapping Event	128	Crystal Report Viewer applet	241
Files		Report Viewer Bean	243
Data Definition	302	Report Viewer Object Model	267
Data Definition, creating	303	Java Bean, Report Viewer	240
.def	355		
Module Definition in C	355	K	
UFJOB in C	356	key combinations	6
FindText method	101	keyboard shortcuts	6
FirstPageButtonClicked Event	257	L	
Format Event (Section Object)	148	LastPageButtonClicked Event	258
Function definition	346	Libraries	
Function definition table	347	Crystal Data Source Type	311, 335
Function examples table	349	UFL, See User-Defined Functions	
Function templates table	348	licensing, runtime	15
G		LogOffServer method	
GetColCount method	331	Report Designer Application Object	18
GetCurrentPageNumber method	248	Report Designer Database Object	44
getEOF method	331	LogOnServer method	
GetFieldData method	332	Report Designer Application Object	19
GetFieldName method	332	Report Designer Database Object	45
GetFields method	245	LogOnServerEx method	
GetFieldType method	333	Report Designer Application Object	20
GetItemByName method		Report Designer Database Object	46
DatabaseFieldDefinitions Collection	50		
FormulaFieldDefinitions Collection	76		
ParameterFieldDefinitions Collection	111		
RunningTotalFieldDefinitions Collection	138		
SQLExpressionFieldDefinitions Collection	157		

M

methods (CRDataSource)	
MoveFirst	338
MoveNext	338
methods (CrystalComObject)	
AddField	329
AddRows	330
DeleteField	331
GetColCount	331
getEOF	331
GetFieldData	332
GetFieldName	332
GetFieldType	333
MoveFirst	333
MoveNext	334
MoveTo	334
Reset	334
methods (JavaBean)	
closeCurrentView	270
exportView	270
printView	271
refreshReport	271
searchForText	271
showLastPage	271
showPage	272
stopAllCommands	272
methods (Report Designer)	
Add	
ConnectionProperties Collection	35
CrossTabGroups Collection	39
DatabaseTables Collection	54
FieldDefinitions Collection	62
FieldElements Collection	67
FormulaFieldDefinitions Collection	76
ObjectSummaryFieldDefinitions Collection	88
ParameterValueInfos Collection	114
ParameterValues Collection	113
ReportAlerts Collection	130
RunningTotalFieldDefinitions Collection	138
Sections Collection	150
SortFields Collection	152
SQLExpressionFieldDefinitions Collection	156
SubreportLinks Collection	158
SummaryFieldDefinitions Collection	164
TableLinks Collection	166
AddADOCCommand, Database Object	43
AddBlobFieldObject, Section Object	141
AddBoxObject, Section Object	141
AddCrossTabObject, Section Object	142
AddCurrentRange, ParameterFieldDefinition Object	107
AddCurrentValue, ParameterFieldDefinition Object	108
AddDefaultValue, ParameterFieldDefinition Object	108

AddFieldObject, Section Object	142
AddGraphObject, Section Object	143
AddGroup, Report Object	120
AddLineObject, Section Object	143
AddOLEDBSource, Database Object	43
AddPictureObject, Section Object	144
AddReportVariable, Report Object	121
AddSpecialVarFieldObject, Section Object	144
AddStoredProcedure, DatabaseTables Collection	55
AddSubreportObject, Section Object	145
AddSummaryFieldObject, Section Object	145
AddTextObject, Section Object	146
AddUnboundFieldObject, Section Object	147
Application Object	17
Area Object	24
AutoSetUnboundFieldSource, Report Object	121
CancelPrinting, Report Object	122
CanClose, Application Object	17
Check	
FormulaFieldDefinition Object	75
SQLExpressionFieldDefinition Object	155
CheckDifferences, DatabaseTable Object	51
ClearCurrentValueAndRange, ParameterFieldDefinition Object	109
ConnectionProperties Collection	35
CreatePageGenerator, PageEngine Object	95
CreateSubreportPageGenerator, PageGenerator Object	99
CrossTabGroups	38
Database Object	43
DatabaseFieldDefinitions Collection	50
DatabaseTable Object	51
DatabaseTables Collection	54
Delete	
ConnectionProperties Collection	36
CrossTabGroups Collection	39
DatabaseTables Collection	56
FieldDefinitions Collection	62
FieldElements Collection	67
FormulaFieldDefinitions Collection	76
ObjectSummaryFieldDefinitions Collection	88
ParameterValueInfos Collection	115
ParameterValues Collection	113
ReportAlerts Collection	131
RunningTotalFieldDefinitions Collection	138
Sections Collection	151
SortFields Collection	153
SQLExpressionFieldDefinitions Collection	156
SubreportLinks Collection	159
SummaryFieldDefinitions Collection	165
TableLinks Collection	167
DeleteAll, ConnectionProperties Collection	37
DeleteGroup, Report Object	122
DeleteNthDefaultValue, ParameterFieldDefinition Object	109

DeleteObject, Section Object	147	ParameterValueInfos Collection.....	114
DiscardSavedData, Report Object	122	ParameterValues Collection.....	113
DrillOnGraph, PageGenerator Object.....	99	PrinterSetup, Report Object.....	124
DrillOnMap, PageGenerator Object	100	PrintOut, Report Object.....	124
DrillOnSubreport, PageGenerator Object	100	PrintOutEx, Report Object	125
Export		PromptForExportOptions, ExportOptions Object	61
PageGenerator Object	101	ReadRecords, Report Object.....	125
Report Object	122	ReimportSubreport, SubreportObject Object	161
ExportOptions Object	61	RenderEPF, Page Object.....	93
FieldElement Object	66	RenderHTML, Page Object.....	94
FieldElements Collection	67	RenderTotalerETF	
FieldObject Object	72	PageEngine Object	96
FindText, PageGenerator Object	101	PageGenerator Object	102
FormulaFieldDefinition Object	75	RenderTotalerHTML	
FormulaFieldDefinitions Collection	75	PageEngine Object	96
GetItemByName		PageGenerator Object	103
DatabaseFieldDefinitions Collection	50	Report Object.....	120
FormulaFieldDefinitions Collection.....	76	ReportAlerts Collection.....	130
ParameterFieldDefinitions Collection.....	111	Reset, ExportOptions Object.....	61
RunningTotalFieldDefinitions Collection.....	138	RunningTotalFieldDefinition Object	135
SQLExpressionFieldDefinitions Collection.....	157	RunningTotalFieldDefinitions Collection	137
GetLicenseStatus, Report Designer Application		SaveAs, Report Object.....	126
Object	18	Section Object.....	140
GetNextRows, Report Object.....	123	Sections Collection.....	150
GetNthCurrentRange, ParameterFieldDefinition		SelectPrinter, Report Object	126
Object	109	SetDataSource	
GetNthCurrentValue, ParameterFieldDefinition		Database Object.....	47
Object	110	DatabaseTable Object	51
GetNthDefaultValue, ParameterFieldDefinition		SetDialogParentWindow, Report Object.....	126
Object	110	SetEvaluateConditionField,	
GetPageNumberForGroup, PageGenerator		RunningTotalFieldDefinition Object	135
Object.....	101	SetInstanceIDField, Area Object	25
GetReportVariableValue, Report Object	123	SetLineSpacing	
GetVersion, Application Object.....	18	FieldElement Object	66
ImportSubreport, Section Object.....	147	FieldObject Object	72
LogOffServer		TextObject Object	170
Application Object	18	SetMatchLogOnInfo, Application Object	22
Database Object	44	SetNoEvaluateCondition,	
LogOnServer		RunningTotalFieldDefinition Object	136
Application Object	19	SetNoResetCondition, RunningTotalFieldDefinition	
Database Object	45	Object	136
LogOnServerEx		SetNthDefaultValue, ParameterFieldDefinition	
Application Object	20	Object	110
Database Object	46	SetOleLocation, OleObject Object.....	92
NewReport, Application Object.....	21	SetParentIDField, Area Object	25
ObjectSummaryFieldDefinitions Collection	87	SetReportVariableValue, Report Object	127
OLEObject Object	91	SetResetConditionField,	
OpenReport, Application Object	21	RunningTotalFieldDefinition Object	136
OpenSubreport		SetSecondarySummarizedField	
Report Object	124	RunningTotalFieldDefinition Object	136
SubreportObject Object.....	161	SummaryFieldDefinition Object	163
Page Object.....	93	SetSummarizedField	
PageEngine Object	95	RunningTotalFieldDefinition Object	137
PageGenerator Object	98	SummaryFieldDefinition Object	163
ParameterFieldDefinition Object	107	SetTableLocation, DatabaseTable Object	52
ParameterFieldDefinitions Collection.....	111	SetText, TextObject Object.....	170

methods (Report Designer) <i>continued</i>	
SetUnboundFieldSource, FieldObject Object	73
SetUserPaperSize.....	127
SortFields Collection.....	152
SQLExpressionFieldDefinition Object.....	155
SQLExpressionFieldDefinitions Collection	156
SubreportLinks Collection.....	158
SubreportObject Object	160
SummaryFieldDefinition Object	163
SummaryFieldDefinitions Collection	164
TableLinks Collection	166
TestConnectivity, DatabaseTable Object	53
TextObject Object.....	170
Verify, Database Object	47
methods (Report Viewer)	
ActivateView, CRViewer Object.....	247
AddParameter, WebReportSource Object.....	264
AddParameterEx, WebReportSource Object	264
AddView, CRViewer Object	247
CloseView, CRViewer Object.....	248
CRVEventInfo Object.....	245
CRViewer Object	247
GetCurrentPageNumber, CRViewer Object.....	248
GetFields, CRVEventInfo Object.....	245
GetViewName, CRViewer Object	248
GetViewPath, CRViewer Object.....	249
JavaBean	270
PrintReport, CRViewer Object.....	249
Refresh, CRViewer Object	250
SearchByFormula, CRViewer Object	250
SearchForText, CRViewer Object	250
ShowFirstPage, CRViewer Object	250
ShowGroup, CRViewer Object.....	250
ShowLastPage, CRViewer Object	251
ShowNextPage, CRViewer Object	251
ShowNthPage, CRViewer Object.....	251
ShowPreviousPage, CRViewer Object	251
ViewReport, CRViewer Object	251
WebReportSource Object.....	264
Zoom, CRViewer Object	252
Module Definition (.def) File in C.....	355
Modules	
Helper	344
UFJOB	350
UFJOB in C	356
MoveFirst method	
CRDataSource	338
CrystalComObject	333
MoveNext method	
CRDataSource	338
CrystalComObject	334
MoveTo method (CrystalComObject).....	334

N

NewReport method	21
NextPageButtonClicked Event.....	258
NoData Event	128

O

Object Model	
ActiveX Report Viewer	243
Crystal Data Objects	328
Crystal Data Source.....	328
CrystalComObject.....	328
Report Designer	14
objects	
naming conflicts with Report Designer	16
passing CRDataSource object to	
Active Data Driver.....	318
Objects (CDO)	
Crystal Data	308, 328
Crystal Data Object Model.....	311
CrystalComObject.....	328
Rowset, <i>see</i> Rowset Object	
Objects (Report Designer).....	16
Application	17
Area	22
BlobFieldObject.....	26
ConnectionProperty	29
CrossTabGroup	37
CrossTabObject	40
Database	42
DatabaseFieldDefinition.....	48
DatabaseTable	50
ExportOptions	56
FieldElement	62
FieldMappingData	67
FormattingInfo.....	73
FormulaFieldDefinition	74
GraphObject.....	77
GroupNameFieldDefinition.....	83
LineObject.....	84
MapObject.....	86
OlapGridObject.....	88
OLEObject.....	90
Page	92
PageEngine	94
PageGenerator	97
ParameterFieldDefinition	104
ParameterValue.....	112
ParameterValueInfo.....	113
PrintingStatus	115
Report	116
RunningTotalFieldDefinition	133
Section	139

SortField	151
SpecialVarFieldDefinition	153
SQLExpressionFieldDefinition	154
SubreportLink	157
SubreportObject	159
SummaryFieldDefinition	162
TableLink	165
TextObject	167
Objects (Report Viewer)	234
CRField	244
CRVEventInfo	244
CRViewer	245
CRVTrackCursorInfo	263
WebReportBroker	263
WebReportSource	263
OnReportSourceError Event	258
OpenReport method	21
OpenSubreport method	
Report Designer Report Object	124
Report Designer SubreportObject Object	161

P

Parameter Blocks	352
Picture Function, sample User-Defined Function	353
PrevPageButtonClicked Event	259
PrintButtonClicked Event	259
PrinterSetup method	124
PrintOut method	124
PrintOutEx method	125
PrintReport method	249
product registration	5
programming	
User-Defined Functions in C	340, 343
User-Defined Functions in Delphi	372
User-Defined Functions in VB	360
PromptForExportOptions method	61
Properties (CrystalComObject), RowCount	329
Properties (Report Designer)	
Area Object	23
Areas Collections	25
BlobFieldObject Object	26
BoxObject Object	28
ConnectionProperties Collection	35
ConnectionProperty Object	30
CrossTabGroup Object	37
CrossTabGroups Collection	38
CrossTabObject	40
Database Object	42
DatabaseFieldDefinition Object	48
DatabaseFieldDefinitions Collection	49
DatabaseTable Object	50
DatabaseTables Collection	53
ExportOptions Object	56
FieldDefinitions Collection	61
FieldElement Object	63

FieldElements Collection	66
FieldMappingData Object	67
FieldObject Object	68
FormattingInfo Object	73
FormulaFieldDefinition Object	74
FormulaFieldDefinitions Collection	75
GraphObject Object	77
GroupNameFieldDefinition Object	83
GroupNameFieldDefinitions Collection	84
LineObject Object	84
MapObject Object	86
ObjectSummaryFieldDefinitions Collection	87
OlapGridObject Object	88
OLEObject Object	90
Page Object	93
PageEngine Object	95
PageGenerator Object	98
Pages Collection	104
ParameterFieldDefinition Object	104
ParameterFieldDefinitions Collection	111
ParameterValue Object	112
ParameterValueInfo Object	114
ParameterValueInfos Collection	114
ParameterValues Collection	112
PrintingStatus Object	115
Report Object	116
ReportAlert Object	129
ReportAlertInstance Object	132
ReportAlertInstances Collection	132
ReportAlerts Collection	130
ReportObjects Collection	133
RunningTotalFieldDefinition Object	133
RunningTotalFieldDefinitions Collection	137
Section Object	139
Sections Collection	150
SortField Object	151
SortFields Collection	152
SpecialVarFieldDefinition Object	153
SQLExpressionFieldDefinition Object	154
SQLExpressionFieldDefinitions Collection	155
SubreportLink Object	157
SubreportLinks Collection	158
SubreportObject Object	159
SummaryFieldDefinition Object	162
SummaryFieldDefinitions Collection	164
TableLink Object	165
TableLinks Collection	166
TextObject Object	168
Properties (Report Viewer)	
CRField Object	244
CRFields Collection	244
CRVEventInfo Object	245
CRViewer Object	245
CRVTrackCursorInfo Object	263
JavaBean	267
WebReportSource Object	264

R

RDC		
features, royalty-required	15	
runtime licensing	15	
RDO data sources	298	
ReadRecords method	125	
Refresh method	250	
RefreshButtonClicked Event	259	
registration		
fax numbers	5	
of product	5	
web site	5	
RelImportSubreport method	161	
RenderEPF method	93	
RenderHTML method	94	
RenderTotallerETF method		
Report Designer PageEngine Object	96	
Report Designer PageGenerator Object	102	
RenderTotallerHTML method		
Report Designer PageEngine Object	96	
Report Designer PageGenerator Object	103	
Report Designer		
Application Object	17	
Application Object methods	17	
Area Object	22	
Area Object methods	24	
Area Object Properties	23	
Areas Collection	25	
Areas Collection Properties	25	
BlobFieldObject Object	26	
BoxObject Object Properties	28	
Collections	16	
ConnectionProperty Object	29	
ConnectionProperty Object Properties	30	
ConnnectionProperties Collection methods	35	
CrossTabGroup Object	37	
CrossTabGroup Object Properties	37	
CrossTabGroups Collection	38	
CrossTabGroups Collection methods	38	
CrossTabGroups Collection Properties	38	
CrossTabObject Object	40	
CrossTabObject Object Properties	40	
Database Object	42	
Database Object methods	43	
Database Object Properties	42	
DatabaseFieldDefinition Object	48	
DatabaseFieldDefinition Object Properties	48	
DatabaseFieldDefinitions Collection	49	
DatabaseFieldDefinitions Collection Methods	50	
DatabaseFieldDefinitions Collection Properties	49	
DatabaseTable Object	50	
DatabaseTable Object methods	51	
DatabaseTable Object Properties	50	
DatabaseTables Collection	53	
DatabaseTables Collection methods	54	
DatabaseTables Collection Properties	53	
Enumerated Types	201	
ExportOptions Object	56	
ExportOptions Object methods	61	
ExportOptions Object Properties	56	
FieldDefinitions Collection	61	
FieldDefinitions Collection methods	62	
FieldDefinitions Collection Properties	61	
FieldElement Object	62	
FieldElement Object methods	66	
FieldElement Object Properties	63	
FieldElements Collection	67	
FieldElements Collection methods	67	
FieldMappingData Object	67	
FieldMappingData Object Properties	67	
FieldObject Object methods	72	
FieldObject object Properties	68	
FormattingInfo Object	73	
FormattingInfo Object Properties	73	
FormulaFieldDefinition Object	74	
FormulaFieldDefinition Object methods	75	
FormulaFieldDefinition Object Object	74	
FormulaFieldDefinitions Collection	75	
FormulaFieldDefinitions Collection methods	75	
FormulaFieldDefinitions Collection Properties	75	
GraphObject Object	77	
GraphObject Object Properties	77	
GroupNameFieldDefinition Object	83	
GroupNameFieldDefinition Object Properties	83	
GroupNameFieldDefinitions Collection	84	
Properties	84	
LineObject Object	84	
LineObject Object Properties	84	
MapObject Object	86	
MapObject Object Properties	86	
Object Model	14	
object naming conflicts	16	
Objects	16	
ObjectSummaryFieldDefinitions Collection	87	
ObjectSummaryFieldDefinitions Collection		
methods	87	
ObjectSummaryFieldDefinitions Collection		
Properties	87	
OlapGridObject Object	88	
OlapGridObject Object Properties	88	
OLEObject Object	90	
OLEObject Object methods	91	
OLEObject Object Properties	90	
Page Object	92	
Page Object methods	93	
Page Object Properties	93	
PageEngine Object	94	
PageEngine Object methods	95	
PageEngine Object Properties	95	
PageGenerator Object	97	

PageGenerator Object methods	98	SQLExpressionFieldDefinition Object.....	154
PageGenerator Object Properties	98	SQLExpressionFieldDefinition Object methods.....	155
Pages Collection	103	SQLExpressionFieldDefinition Object Properties.....	154
Pages Collection Properties.....	104	SQLExpressionFieldDefinitions Collection	155
ParameterFieldDefinition Object	104	SQLExpressionFieldDefinitions Collection methods.....	156
ParameterFieldDefinition Object methods	107	SQLExpressionFieldDefinitions Collection Properties.....	155
ParameterFieldDefinition Object Properties	104	SubreportLink Object	157
ParameterFieldDefinitions Collection.....	111	SubreportLink Object Properties	157
ParameterFieldDefinitions Collection methods.....	111	SubreportLinks Collection.....	158
ParameterFieldDefinitions Collection Properties.....	111	SubreportLinks Collection methods	158
ParameterValue Object.....	112	SubreportLinks Collection Properties	158
ParameterValue Object Properties	112	SubreportObject Object	159
ParameterValueInfo Object.....	113	SubreportObject Object methods	160
ParameterValueInfo Object Properties	114	SubreportObject Object Properties	159
ParameterValueInfos Collection.....	114	SummaryFieldDefinition Object	162
ParameterValueInfos Collection methods.....	114	SummaryFieldDefinition Object methods	163
ParameterValueInfos Collection Properties	114	SummaryFieldDefinition Object Properties	162
ParameterValues Collection	112	SummaryFieldDefinitions Collection	164
ParameterValues Collection methods.....	113	SummaryFieldDefinitions Collection methods.....	164
ParameterValues Collection Properties	112	SummaryFieldDefinitions Collection Properties	164
PrintingStatus Object	115	TableLink Object Object	165
PrintingStatus Object Properties.....	115	TableLink Object Properties	165
Report Object.....	116	TableLinks Collection	166
Report Object Events	127	TableLinks Collection methods.....	166
Report Object methods.....	120	TableLinks Collection Properties.....	166
Report Object Properties.....	116	TextObject Object	167
ReportAlert Object Properties	129	TextObject Object methods.....	170
ReportAlertInstance Object Properties	132	TextObject Object Properties	168
ReportAlertInstances Collection Properties.....	132	Report Viewer	
ReportAlerts Collection methods.....	130	ActiveX.....	233
ReportAlerts Collection Properties	130	ActiveX Object Model.....	243
ReportObjects Collection.....	133	adding Java Bean.....	241
ReportObjects Collection Properties	133	adding to VB project.....	234
RunningTotalFieldDefinition Object	133	application development	232
RunningTotalFieldDefinition Object methods	135	closeCurrentView method, Java Bean	270
RunningTotalFieldDefinition Object Properties	133	connecting to Web Reports Server.....	239
RunningTotalFieldDefinitions Collection	137	controlling appearance.....	238
RunningTotalFieldDefinitions Collection methods	137	CREventInfo Object	244
RunningTotalFieldDefinitions Collection Properties.....	137	CREventInfo Object Properties.....	245
Section Object	139	CRField Object.....	244
Section Object Events	148	CRField Object Properties	244
Section Object methods.....	140	CRFields Collection	244
Section Object Properties	139	CRFields Collection Properties.....	244
Sections Collection	150	CREventInfo Object methods	245
Sections Collection methods.....	150	CRViewer Object	245
Sections Collection Properties.....	150	CRViewer Object Events.....	252
SortField Object.....	151	CRViewer Object methods	247
SortField Object Properties	151	CRViewer Object Properties	245
SortFields Collection	152	CRVTrackCursorInfo Object	263
SortFields Collection methods.....	152	CRVTrackCursorInfo Object Properties.....	263
SortFields Collection Properties	152	Enumerated Types	265
SpecialVarFieldDefinition Object	153	events.....	236
SpecialVarFieldDefinition Object Properties	153	Events, Java Bean.....	272

exportView method, Java Bean.....	270
Java applet.....	241
Java Bean	240, 243
methods, Java Bean	270
moving through report.....	237
printing reports.....	238
printView method, Java Bean	271
Properties, Java Bean	267
refreshReport method, Java Bean	271
searchForText method, Java Bean	271
secure data in reports	235
ServerRequestEvent, Java Bean	272
showLastPage method, Java Bean	271
showPage method, Java Bean	272
specifying report.....	235
stopAllCommands method, Java Bean	272
using in applications.....	231
ViewChangeEvent, Java Bean	272
WebReportBroker Object	263
WebReportSource Object.....	263
WebReportSource Object methods.....	264
WebReportSource Object Properties	264
reports	
and secure data in Crystal Report Viewer	235
moving through with Crystal Report Viewer	237
printing with Crystal Report Viewer	238
specifying with Crystal Report Viewer	235
Reset method	
CrystalComObject	334
Report Designer ExportOptions Object.....	61
RowCount property.....	329
Rowset Object, adding fields	309
royalty-required features (RDC).....	15
runtime code	
ProgIds for CreateObject	
function	17, 112, 113, 114
runtime licensing	15
S	
sample applications	
Embeddable Designer in Visual Basic.....	274
Embeddable Designer in Visual C++	280
sample code	
Crystal Report Viewer in VB Script	240
Crystal Report Viewer in	
Visual Basic	235, 236, 237, 238
RDC in Visual Basic	
Add method (ReportAlerts collection)	131
GetVersion method.....	18
SaveAs method	126
SearchButtonClicked Event	260
SearchByFormula method	250
SearchExpertButtonClicked Event.....	260
SearchForText method	250
SelectionFormulaBuilt Event	260

SelectionFormulaButtonClicked Event	261
SelectPrinter method.....	126
SetDataSource method	
Report Designer Database Object	47
Report Designer DatabaseTable Object.....	51
SetDialogParentWindow method.....	126
SetEvaluateConditionField method	135
SetInstanceIDField method	25
SetLineSpacing method	
Report Designer FieldElement Object.....	66
Report Designer FieldObject Object	72
Report Designer TextObject Object	170
SetMatchLogOnInfo method	22
SetNoEvaluateCondition method	136
SetNoResetCondition method	136
SetNthDefaultValue method	110
SetOleLocation method	92
SetParentIDField method	25
SetReportVariableValue method	127
SetResetConditionField method	136
SetSecondarySummarizedField method	
RunningTotalFieldDefinition Object.....	136
SummaryFieldDefinition Object	163
SetSummarizedField method	
Report Designer RunningTotalFieldDefinition	
Object	137
Report Designer SummaryFieldDefinition	
Object	163
SetTableLocation method	52
SetText method.....	170
SetUnboundFieldSource method	73
SetUserPaperSize method	127
shortcuts, keyboard.....	6
ShowFirstPage method.....	250
ShowGroup Event.....	261
ShowGroup method	250
ShowLastPage method.....	251
ShowNextPage method.....	251
ShowNthPage method	251
ShowPreviousPage method.....	251
StopButtonClicked Event.....	261
subreports, reimporting.....	161
support	
product registration	5
technical	5, 6
web site	5, 6

T

tables	
error in C	349
Function definition in C	347
Function example in C	349
Function template in C.....	348
technical support.....	5, 6

TermForJob	350
implementing	357
see also User-Defined Functions, programming in C	
TestConnectivity method	53
tutorials	
Visual Basic, creating an application with the	
Embeddable Designer	274
Visual C++, creating an application with the	
Embeddable Designer	280
Type Library, Crystal Data Source	335

U

UFEndJob	369, 379
See also User-Defined Functions,	
programming in Delphi	
See also User-Defined Functions,	
programming in VB	
UFLInitialize	368, 379
See also User-Defined Functions,	
programming in Delphi	
See also User-Defined Functions,	
programming in VB	
UFJOB	356
UFL, See User-Defined Functions	
UFStartJob	368, 379
See also User-Defined Functions,	
programming in Delphi	
See also User-Defined Functions,	
programming in VB	
UFTerminate	368, 379
See also User-Defined Functions,	
programming in Delphi	
See also User-Defined Functions,	
programming in VB	
User Function Libraries, See User-Defined Functions	
User-Defined Errors, See User-Defined Functions, errors	
User-Defined Functions	
arrays in Delphi	376
arrays in VB	366
data types in C	341
data types in Delphi	376
data types in VB	365
error table in C	349
errors	
in C	351
in Delphi	378
in VB	367
Function definition in C	346
Function definition table in C	347
Function examples table in C	349
function name prefixing	
in Delphi	377
in VB	366
Function templates table in C	348
Helper Modules in C	344

implementing in C	351
in C	339
in Delphi	371
in VB	359
Module Definition (.def) File in C	355
naming	
in C	340
in VB	366
overview	
in C	340
in Delphi	372
in VB	360
passing parameters	
in C	352
in Delphi	378
in VB	367
programming	
in C	340
in Delphi	372
in VB	360
the UFL in C	343
reserved names in Delphi	377
return types in C	342
sample Automation Server in VB	369
sample in C	353
setting up a UFL project in C	345
special purpose functions	
in C	350, 357
in Delphi	378
in VB	368
UFJOB modules in C	356
UFL states in C	345
using in VB	364

V

Verify method	47
ViewChanged Event	262
ViewChanging Event	262
ViewReport method	251
Visual Basic	
adding Crystal Report Viewer to project	234
ProgIds for CreateObject	
function	17, 112, 113, 114
Visual Basic, See also User-Defined Functions	

W

Web Reports Server, connecting to Report Viewer ..	239
web sites	
consulting	6
training	5

Z

Zoom method	252
ZoomLevelChanged Event	263