Crystal Reports™ 8.5
Technical Reference Guide

Seagate Software IMG Holdings, Inc.
915 Disc Drive
Scotts Valley
California, USA 95066

Documentation Issue 1. Febuary 2001.

Contents

Chapter 7: Active Data

Chapter 8: Crystal Data Source Object Models

Chapter 9: The Crystal Active Data Driver Reference

Chapter 10: Creating User-Defined Functions in C

Visual Basic Solutions 1

Crystal Reports offers you a wide range of solutions for your reporting needs. This chapter provides a general overview of the more common Crystal Reports development tools. These include the Crystal Report Engine API, Crystal ActiveX Control, and Crystal Report Engine Automation Server, but not the Report Designer Component. The Report Designer Component is the preferred Development tool and is described in previous chapters.

Enhancements to the Crystal Report Print Engine API

There are many enhancements to the API that correspond to enhancements in Version 8 and 8.5 of Crystal Reports. For a more detailed description of the enhancements that follow, refer to the documentation for the related functionality in the *Crystal Reports User's Guide*. For developer reference information on each of the new and enhanced calls, see the *Crystal Reports Developer's Help (CrystalDevHelp.chm)* or the *Crystal Reports Technical Reference Guide*.

Many new API structures and functions have been introduced and many existing ones have been enhanced, as outlined in the following sections:

- "Version 8.5 enhancements" on page 2
- "Version 8 enhancements" on page 3

Version 8.5 enhancements

The following enhancements apply to version 8.5 of Crystal Reports.

- "Create, modify, and monitor Report Alerts" on page 2
- "Get report version information" on page 3
- "New additions to the PEReportOptions structure" on page 3
- "Additional error codes" on page 3

Create, modify, and monitor Report Alerts

Report Alerts are custom messages created in the Crystal Reports Designer that appear when certain conditions are met by data in a report. 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 Guide*.

Two new structures and seven new functions have been added to the Crystal Report Engine API to allow you to create, modify, and monitor Report Alerts at runtime. These structures and their functions are:

- "PEAlertInstanceInfo" on page 453
- "PEGetNthAlertInstanceInfo" on page 333
- "PEReportAlertInfo" on page 494
- "PEGetNReportAlerts" on page 328
- "PEEnableNthAlert" on page 296
- "PEGetNthReportAlert" on page 343
- "PESetNthAlertConditionFormula" on page 418
- "PESetNthAlertDefaultMessage" on page 419
- "PESetNthAlertMessageFormula" on page 420

Get report version information

In the CRPE API you may now get the report's version information. A new structure, "PEVersionInfo" on page 518, has been added to hold the major number, minor number, and letter information. The structure is returned by calling the newly added function, "PEGetReportVersion" on page 359.

New additions to the PEReportOptions structure

In the CRPE API, three new elements are now supported in the structure, "PEReportOptions" on page 496:

- alwaysSortLocally
- isReadonly
- canSelectDistinctRecords

You may use the CRPE API calls, "PEGetReportOptions" on page 357 and "PESetReportOptions" on page 436, to obtain or set report options.

Additional error codes

The CRPE API now supports three new error codes. These error codes are:

- PE_ERR_READONLYPARAMETEROPTION
- PE_ERR_MINGREATERTHANMAX
- PE_ERR_INVALIDSTARTPAGE

For more information see "Error Codes" on page 545.

Version 8 enhancements

The following enhancements apply to version 8 of Crystal Reports.

Launch Seagate Analysis

You may now have users click a button on the preview window toolbar to launch Seagate Analysis and open the current report.

A new event, LaunchSeagateAnalysisEvent, has been added. If you enable the event, and the user clicks the Launch Seagate Analysis button, the LaunchSeagateAnalysisEvent will be fired. The user will be prompted to save the file and Seagate Analysis will be launched with the current report open.

There are a number of other enhancements in the CRPE API that complement those in corresponding components of the Crystal Report Designer.

Basic and Crystal Syntax support

Since the Crystal Reports formula language has been enhanced to support a Basic-like syntax as well as the Crystal syntax, existing Formula API calls have been modified accordingly.

In all the Formula calls now, SET uses the syntax specified by the user and GET sets a flag that the user can later retrieve.

Two new API calls, PESetFormulaSyntax and PEGetFormulaSyntax, have also been added.

Charting enhancements

The following are several enhancements for charting:

- Automatic scaling for axes
- Default titles
- Legend layout
- Fractional point size

Automatic scaling for axes

In the CRPE API, three new elements are now supported in the structure, PEGraphAxisInfo:

- dataAxisYAutoScale
- dataAxisY2AutoScale
- seriesAxisAutoScale

You may use the CRPE API calls, PEGetGraphAxisInfo and PESetGraphAxisInfo, to obtain or set automatic scaling for the graph axes.

Default titles

The Crystal Report Designer, by default, now creates title, subtitle, footnote, X-axis title, Y-axis title and Z-axis title. In the CRPE API, you can use PEGetGraphTextDefaultOption and PESetGraphTextDefaultOption to manipulate these items. To use these calls, you will need to specify a PE_GTT_* constant that corresponds to each of the titles, and set useDefault as TRUE or FALSE.

Legend layout

In the CRPE API, you may now set the legendLayout element in PEGraphOptionInfo structure to one of the PE_GLL_* constants. You may also use the CRPE API calls, PEGetGraphOptionInfo and PESetGraphOptionInfo, to get or set legend layout options.

Fractional point size

For fonts, there is now support for fractional point sizes including the 10.5 point font size commonly used in Japanese. A new twipSize element has been added to the structure PEFontColorInfo. Related functions and methods now return or apply the twipSize element if the pointSize element is zero. The functions affected are:

- PEGetGraphFontInfo
- PESetGraphFontInfo

Hierarchical grouping

In the Crystal Report Designer, you may now use hierarchical grouping to arrange data in a report to show hierarchical relationships in your data.

To support hierarchical grouping, the PEGroupOptions structure and the PEGetGroupOptions and PESetGroupOptions calls in the CRPE API have been modified.

Hyperlinks & On-demand subreports

The CRPE API has been enhanced to support hyperlinks and on-demand subreports.

New elements have been added to the PETrackCursorInfo structure to support these enhancements:

- short ondemandSubreportCursor – this cursor can be set to display over on-demand subreports when drilldown for the window is enabled; the default is PE_TC_MAGNIFY_CURSOR.
- short hyperlinkCursor – this cursor can be set to display over any report object that contains hyperlink text; the default is PE_TC_HAND_CURSOR.
- new cursor types:
  - PE_TC_BACKGROUND_PROCESS_CURSOR
  - PE_TC_GRAB_HAND_CURSOR
  - PE_TC_ZOOM_IN_CURSOR
  - PE_TC_REPORT_SECTION_CURSOR
  - PE_TC_HAND_CURSOR

Using the Crystal Report Engine API in Visual Basic

This section provides additional information for developers working in Visual Basic. Several features of the Crystal Report Engine must be handled differently in Visual Basic than in other development environments. In addition, some of the topics here are designed to simply assist Visual Basic programmers in the design of applications using the Crystal Report Engine.

The following topics are discussed in this section:

When to Open/Close the Crystal Report Engine

In a Visual Basic application, you can either open the Crystal Report Engine when you open your application or when you open a form. As a general rule, it is always best to open the Crystal Report Engine when you open the application and close it when you close the application. Here is why:

- When you open and close a form, the Crystal Report Engine opens every time you open the form and closes every time you close the form. If you print a report, close the form, and later decide to print a report again, the application has to reopen the Crystal Report Engine when you open the form, creating a time delay while running your application.
- When you open and close the application, the Crystal Report Engine opens as you start the application, and stays open as long as the application is open. Once the Crystal Report Engine is open, you can print a report as often as you wish without the need to reopen the Crystal Report Engine every time you print.

Embedded Quotes in Visual Basic Calls to the Crystal Report Engine

When you pass a concatenated string from Visual Basic to the Crystal Report Engine (for example, for a record selection formula), it is important that the resulting string has the exact syntax that the Crystal Report Engine expects. You should pay special attention to embedded quotes in concatenated strings because they are often the source of syntax errors.

Several examples follow. The first example shows code with a common embedded quote syntax error and the last two examples show code using the correct syntax.

Incorrect syntax

```
VBNameVariable$ = "John"
Recselct$ = "{file.LASTNAME} = " + VBNameVariable$
```

This code results in the string:

```
{file.LASTNAME} = John
```

Since John is a literal string, the Formula Checker expects to see it enclosed in quotes. Without the quotes, the syntax is incorrect.

Correct syntax

```
VBNameVariable$ = “John”
Recselct$ = “{file.LASTNAME} = “ +
(chr$(39) + VBNameVariable + chr$(39)
```

This code results in the string:

```
{file.LASTNAME} = 'John'
```

This is the correct syntax for use in a Crystal Reports record selection formula. This is the syntax you would use if you were entering a selection formula directly into Crystal Reports.

```
VBNameVariable$ = “John”
Recselct$ = “{file.Lastname} = “
(+ “'” + VBNameVariable + “'”
```

This code also results in the string:

```
 {file.LASTNAME} = 'John'
```

Again, the correct syntax.

Passing Dates/Date Ranges in Visual Basic using the Crystal Report Engine API Calls

You may want to pass date or date range information from your Visual Basic application to the Crystal Report Engine for use in formulas, selection formulas, etc. Here is an example showing a way to do it successfully:

1. Start by opening a print job and assigning the print job handle to a variable.

```
JobHandle% = PEOpenPrintJob (“C:\CRW\CUSTOMER.RPT”)
```

2. Create variables that hold the year, month, and day for both the start and end of the range.

```
StartYear$ = 1992
StartMonth$ = 01
StartDay$ = 01
EndYear$ = 1993
EndMonth$ = 12
EndDay$ = 31
```

3. Now build a string to pass to the record selection formula. This is done in two steps:

- First, build the starting and ending dates for the date range.
  - Assign the starting date string to the variable StrtSelect$.
  - Assign the ending date string to the variable EndSelect$.

```
StrtSelect$ = "{filea.STARTDATE} < Date
(" + StartYear$ + ", " + StartMonth$ + ", "
+ StartDay$ +")"
EndSelect$ = "{filea.ENDDATE} < Date
(" + EndYear$ + ", " + EndMonth$ +
", " + EndDay$ +")"
```

- Second, build the selection formula using the StrtSelect$ and EndSelect$ variables.

```
Recselct$ = StrtSelect$ + " AND " + EndSelect$
```

4 Once your formula is built, set the record selection formula for the report.

```
RetCode% = PESetSelectionFormula
(JobHandle%, RecSelect$)
```

5 Finally, print the report.

```
RetCode% = PEStartPrintJob (JobHandle, 1)
RetCode% = PEClosePrintJob (JobHandle, 1)
```

6 Modify this code to fit your needs.

Identifying String Issues in Visual Basic Links to the Crystal Report Engine

When passing a string to the Crystal Report Engine as part of the Custom-Print Link, you may think that you are passing one thing when the program, in fact, is passing something entirely different. This can happen easily, for example, when you are passing a concatenated string that you have built using variables. A small syntax error (with embedded quotes, for example) can lead to an error message and a failed call. A simple debugging procedure follows.

To Identify a String Issue (bug)

To identify a string bug, have the program display what it is passing in a message box. To do so, put a line of code similar to the following immediately after the call in question:

```
MsgBox (variablename)
```

Look at the string that is displayed and make certain that it is exactly what Crystal Reports expects for a string.

- If the syntax is incorrect, look for errors in the concatenated string you have built.
- If the syntax is correct, look for other problems that could have caused the call to fail.
- If you are not sure if the syntax is correct, write down the string from the message box, enter it in the Crystal Reports Formula Editor, and click the *Check* button. If there is an error in the string, the Formula Checker will identify it for you.

Hard-coded Nulls in Visual Basic User Defined Types

When you assign a string to a user defined type in Visual Basic, it is necessary to hard-code a null immediately after the string. For example:

```
myStruct.stringField = "Hello" + CHR$(0)
```

Visual Basic Wrapper DLL

Some of the features of the Crystal Report Engine API are not directly available to Visual Basic programmers, due to restrictions in the Visual Basic language, while others present technological issues that are better handled differently from what was originally designed in the Report Engine API. To avoid problems calling functions in the API, you may want to consider using the "Crystal ActiveX Controls" on page 10, or the "Crystal Report Engine Automation Server" on page 12. However, if you prefer to work with the API, Crystal Reports includes the Visual Basic Wrapper DLL, CRWRAP32.DLL.

The Visual Basic Wrapper DLL has been designed specifically for programming in the Visual Basic environment and can be used to build Crystal Report Engine applications in Visual Basic 4.0 or later. The CRWRAP.BAS module, installed by default in the \Seagate Software\Crystal Reports directory, redefines many of the functions and structures defined in GLOBAL.BAS. When working with the Crystal Report Engine API, add both modules to your Visual Basic project.

The functions and structures defined in the Visual Basic Wrapper DLL provide an interface for handling export formats, parameter fields, SQL server and ODBC logon information, graphs, printing, and more. For complete information on each of the structures and functions included, search for *Visual Basic Wrapper for the Crystal Report Engine* in the *Crystal Reports Developer's Help (CrystalDevHelp.chm)*. In most cases, each function or structure has a corresponding function or structure in the original Crystal Report Engine API with a similar name. When working in Visual Basic though, you must use the functions and structures provided by the Visual Basic Wrapper DLL.

CRPE32.DEP

The crpe32.dep dependency file contains only the most often used runtime files for Visual Basic. The file can be modified to include any runtime file. To include a file that is not being distributed when using the existing crpe32.dep, you may add the new file to the "Additional Runtime DLLs" section. However, you must continue a sequential order to the Uses=.

For example:

If you want to include the p2BACT.DLL, you must add it under the "Additional Runtime DLLs" section of crpe32.dep.

If the last numbered Uses= is:

```
Uses46=\program files\seagate software\sschart\SSCSDK32.DLL
```

To include the p2BACT.DLL, add the following line to crpe32.dep:

```
Uses47=..\crystal\p2BACT.DLL
```

Crystal ActiveX Controls

ActiveX controls bring more powerful applications to desktops and networks. ActiveX moves beyond applications that produce static documents to a Windows environment that provides active controls, documents, and client applications that can operate and interact not only with each other, but also with network intranets and the global Internet.

ActiveX controls provide plug-in capabilities that let you add application components, and even entire applications, to your own development projects without writing a line of code. Crystal Reports includes the Crystal ActiveX Control. Use the ActiveX Control to easily add all of the report processing power of Crystal Reports to your own Visual Basic, Visual C++, Borland C++, Delphi, and other applications.

Note:

- The development tools may refer to an ActiveX Control by any of the following names: OLE Control, OCX Control, Custom Control, or ActiveX Control. As long as the term used refers to a control with an .OCX filename extension, it is synonymous with the term ActiveX Control used here.

The following topics are discussed in this section:

Adding the ActiveX Control to your Project

This section demonstrates how to add the Crystal ActiveX Control to an application project being designed in Visual Basic versions 5.0 and 6.0. If you wish to use the ActiveX Control in a different development environment or a different version of Visual Basic, please refer to the documentation that came with your development tools for information on adding an ActiveX or OLE Control (OCX) to your project.

The Crystal ActiveX Control was installed in the \WINDOWS\SYSTEM directory when you installed Crystal Reports. You add the ActiveX Control to your Visual Basic project using the Components command on the Visual Basic Project menu.

1. Open Visual Basic.
2. Open the project to which you want to add the ActiveX Control.
3. Choose **Components** from the Project menu.

   The Components dialog box appears.

- If Crystal Report Control appears in the Available Controls list, click the check box next to it, click **OK**, and skip to Step 6.
- If Crystal Report Control does not appear in the Available Controls list, click **Browse**. The Add ActiveX Control dialog box appears.

Note: Crystal Report Control is the name of the Crystal ActiveX Control when it is added to a development project. The term ActiveX Control refers to a type of control, while Crystal Report Control is the name of the ActiveX Control provided by Crystal Reports.

4. Use the controls in the Add ActiveX Control dialog box to locate and select the CRYSTL32.OCX file. This file was installed in your \WINDOWS\SYSTEM directory when you installed Crystal Reports. Once you locate and select the file, click Open.
5. Crystal Report Control will now appear in the Available Controls list box. Click the check box next to the name of the control, and click OK.

   Visual Basic adds the Crystal ActiveX Control to your toolbox. The tool looks like this:

6. To add the ActiveX Control to a form, double-click the tool in the toolbox and Visual Basic installs it on the active form.

Note: For instructions on how to add an ActiveX Control or OLE control to development applications other than Visual Basic, refer to the documentation that came with the development application you are using.

Using the ActiveX Controls

Once you have the ActiveX Control object on your form, you build the connection between your application and Crystal Reports by setting the object's properties at design time or changing properties at runtime. The ActiveX properties let you specify:

- the name of the report you want to print in response to an application event.
- the destination for that report (window, file, or printer).
- the number of copies you want to print (if your report is going to the printer).
- print file information (if your report is going to a file).
- preview window sizing and positioning information (if your report is going to a window).
- selection formula information (if you want to limit the records in your report).
- sorting information.
- other related properties.

Crystal ActiveX Control properties can be changed either at design time or at runtime. Note, however, some properties are available only at runtime. These properties do not appear in the Properties list at design time.

Note: For a complete description of each property in the Crystal ActiveX Control, refer to the *Crystal ActiveX Control Reference* in the *Techrefvol2.pdf*.

Changing Properties for the ActiveX Control

1. Click the ActiveX control on your form to select it.
2. Right-click and choose **Crystal Properties** from the shortcut menu.
   The Property Pages dialog box appears.
3. Use the tabs and controls in this dialog box to change the ActiveX Control properties at design time.

Note: ActiveX Control properties also appear in the Visual Basic *Properties* list box. For instructions on using the Properties list box, refer to your Visual Basic documentation.

Changing Properties at Runtime

You can set most of the ActiveX Control properties at runtime by adding simple entries to your procedure code. Runtime property settings replace settings you make via the Properties list at design time.

Use the "Action" on page 1252 property or the "PrintReport" on page 1335 method to actually process the report at runtime. The Action property and the PrintReport method can only be used at runtime, and are the only means by which a report can actually be generated by the ActiveX Control.

For information on how to set ActiveX Control properties at runtime, refer to their syntax by searching for each property by name in the *Crystal Reports Developer's Help (CrystalDevHelp.chm)*. Included are examples of how to set each property at runtime.

Crystal Report Engine Automation Server

The Crystal Report Engine Automation Server has been designed as both an object-oriented approach to adding Crystal Report Engine features to your applications, and as an ideal method for displaying reports in web pages. If you work in a development environment that supports access to COM-based automation servers, such as Visual Basic, you will quickly make full use of the Crystal Report Engine Automation Server to add powerful reporting to your applications. In addition, if you manage a web server that supports Active Server Pages, such as Microsoft's Internet Information Server (IIS) or Personal Web Server, the Crystal Report Engine Automation Server satisfies all of your dynamic reporting needs.

The Crystal Report Engine Automation Server (CPEAUT32.DLL) was installed in your \WINDOWS\SYSTEM directory when you installed Crystal Reports. The Crystal Report Engine Automation Server is an in-process automation server based on the Component Object Model (COM). This automation server provides an IDispatch interface, but is not programmable through a vtable interface. For Visual Basic programmers and in Active Server Pages, handling the IDispatch interface is almost transparent. For more information on the Component Object Model and COM interfaces, refer to Microsoft documentation.

The following topics are discussed in this section:

Adding the Automation Server to your Visual Basic Project

Before you can use the Crystal Report Engine Automation Server with a Visual Basic project, it must be registered on your system, and you must add it to your project. If you selected to install Development Tools when you installed Crystal Reports, the automation server will have already been registered on your system. If you did not select Development Tools, run the Crystal Reports setup application again, select Custom installation, and make sure Development Tools are installed.

Note: The following procedures demonstrate the use of the Report Engine Automation Server in versions 5.0 and later of Visual Basic. For information on using automation servers in earlier versions of Visual Basic, or in other development environments, please refer to the documentation that came with your software.

To add the automation server to a project in Visual Basic versions 5.0 or 6.0:

1. With your project open in Visual Basic, choose **References** from the Project menu. The References dialog box appears.

   Note: For complete information on adding ActiveX components to a project, refer to your Visual Basic documentation.

2. The Available References list box shows all available component object libraries currently registered on your system. Scroll through this list box until you find the *Crystal Report Engine 8 Object Library*. This is the Crystal Report Engine Automation Server.

   Note: If the Crystal Report Engine Object Library does not appear in the Available References list box, use the Browse button to locate and select the Crystal Report Engine Automation Server (CPEAUT32.DLL) in your \WINDOWS\SYSTEM directory.

3. Toggle on the check box next to the Crystal Report Engine 8 Object Library reference. This makes the Crystal Report Engine Automation Server available to your project.

4. Click **OK** in the References dialog box.

Using the Automation Server in Visual Basic

There are five primary steps to using the Crystal Report Engine Automation Server in your Visual Basic project:

Creating an Application Object

The Application object in the Crystal Report Engine Automation Server's object library is the only object that can be created. Using the Application object, you can obtain a report object by opening a report file, manipulate aspects of the report object, such as select formulas and sort fields, then print or export the report.

Since the Application object is the only creatable object exposed by the Crystal Report Engine Automation Server, you must create an Application object before you can perform any other tasks using the Crystal Report Engine. Use code similar to the following to create an Application object in your Visual Basic project:

```
Dim app As CRPEAuto.Application
Set app = CreateObject("Crystal.CRPE.Application")
```

Alternately, you can use the following code:

```
Dim app as New CRPEAuto.Application
```

Crystal.CRPE.Application is the Application object's ProgID (programmatic identifier). Visual Basic uses this ID to create an instance of the Application object, which can then be used to obtain a Report object. For a complete description of the CreateObject function, refer to your Visual Basic documentation.

Obtaining a Report Object

You obtain a Report object by specifying a Crystal Reports (.RPT) file and opening it with the OpenReport method of the Application object:

```
Dim report As CRPEAuto.Report
Set report = app.OpenReport("c:\reports\xtreme.rpt")
```

The OpenReport method has only one parameter, the path of the report file you want to access. By setting a report object according to the return value of this method, you can proceed to manipulate, print, preview, or export the report using other objects, methods, and properties available in the Crystal Report Engine Automation Server's object library.

Using the Report Object

Once you obtain a Report object, you can use that object to make runtime changes to the report file, then send the report to a printer, a preview window, a disk file, an e-mail address, an ODBC data source, or a group folder in Microsoft Exchange or Lotus Notes. Note that the changes you make at runtime are not permanent; they do not change the original report file, they only affect the output of the report during the current Crystal Report Engine session.

Through the report object, you obtain access to different aspects of the report file, such as selection formulas, subreports, sort fields, and format settings. For example, the following code changes the record selection formula for the report:

```
report.RecordSelectionFormula = "{customer.Region} = 'CA'"
```

Refer to the reference section of this manual for complete information on all objects, properties, and methods available in the object library and how to use them.

Once you make all desired changes and review settings for the report using the functionality available in the automation server, you can print, preview, or export the report just as you do from Crystal Reports. The automation server provides default settings for these activities, or you can specify your own settings. The simplest technique for sending the report to a printer would look like this:

```
report.PrintOut
```

Without receiving any parameters, the PrintOut method simply sends the report to the default printer with default settings. For more information about methods for the Report object, search for each method by name in *Crystal Reports Developer's Help (CrystalDevHelp.chm)*.

Releasing Objects

Visual Basic will clean up any objects that have not been released when your application terminates. However, since objects use memory and system resources that cannot be accessed by other running applications, you should get into the habit of releasing any objects when you are finished with them.

To release an object, simply set it equal to Nothing:

```
Set report = Nothing
Set app = Nothing
```

Handling Errors

Error trapping for the Crystal Report Engine Automation Server can be handled just like normal error trapping in Visual Basic. When an error occurs in the automation server, the error is sent to Visual Basic which sets the properties of the Err object appropriately. To avoid runtime problems, you should trap for errors inside your Visual Basic code. A typical error trapping routine might look something like this:

```
On Error GoTo HandleError
' Several lines of
' Visual Basic code
HandleError:
    If (Err.Number <> 0) Then
        MsgBox (Err.Description)
    End If
```

The advantage of handling automation server errors like this is that they can be handled at the same time other Visual Basic errors are handled, making your code more efficient and easier for other developers to understand.

Object Name Conflicts

Some object names in the Crystal Report Engine Object Library may conflict with object names in other object libraries attached to your Visual Basic projects. For instance, if your project includes the Data Access Objects (DAO) Object Library, the DAO Database object can conflict with the Report Engine Object Library's Database object. Another common name conflict can occur between the Report Engine's OLEObject and the RichTextLib OLEObject control. Such name conflicts can produce errors in your applications.

Note: RichTextLib is a component included with some versions of Visual Basic.

To avoid name conflicts, you should append all references to Crystal Report Engine Object Library object names with CRPEAuto, the name of the object library as it appears in Visual Basic. For instance, the following code can be used to create a Report object:

```
Dim rpt As CRPEAuto.Report
Set rpt = app.OpenReport("c:\reports\xtreme.rpt")
```

Object names in other object libraries should also be appended with an object library name. For instance, the DAO Database object could be appended with DAO:

```
Dim db As DAO.Database
```

Viewing the Crystal Report Engine Object Library

The Visual Basic Object Browser allows you examine the classes, methods, and properties exposed by any ActiveX component available to your project. If you have selected the Crystal Report Engine Object Library using the References dialog box (see "Adding the Automation Server to your Visual Basic Project" on page 13), then you can browse through the Object Library using the Visual Basic Object Browser:

1. With your project open in Visual Basic, choose **Object Browser** from the View menu. The Object Browser appears.
2. From the Libraries/Projects drop-down list, select the *Crystal Report Engine Object Library*. Classes, methods, and properties exposed by the Object Library will appear in the Object Browser.
3. Select a class in the Classes/Modules list box to view its methods and properties in the Methods/Properties list box.

Note: While viewing the Crystal Report Engine Object Library in the Visual Basic Object Browser, you may notice several classes, methods, and properties that are not documented in the Crystal Reports Technical Reference. There are several features in the Crystal Report Engine Automation Server that are not available with Crystal Reports, and are protected by a security feature built into the Object Library. These features will become available in future Seagate Software products. Contact Seagate Software's Sales department for further information.

Crystal Reports also provides the Crystal Report Engine Object Library Browser Application as a convenient utility for accessing online information about the Crystal Report Engine Object Library. Simply choose the Xtreme Mountain Bike option in the Sample Files when installing, or choose an automatic installation (the files will be installed by default) to install the utility, then browse through the Object Library using the tree control. Select a class, method, or property for more information on how to use it.

Handling Preview Window Events

The Report and Window objects in the Crystal Report Engine Object Library include several Events. By handling these events in your Visual Basic project, you can customize how your application responds to user actions. For instance, if a user clicks on a button in the toolbar of the preview window, such as the Zoom button or the Next Page button, your application can respond to that event.

Note: Events are only available in Visual Basic 5.0 and later. If you are using a version of Visual Basic earlier than 5.0, you will not be able to make use of the Events exposed by the Report or Window object.

To handle Events for the Report or Window object, you must declare the instance of the object as *Public* and *WithEvents*. For example:

```
Public WithEvents repEvents As CRPEAuto.Report
Public WithEvents wndEvents As CRPEAuto.Window
```

Once declared, the objects will appear in the Visual Basic Object window. If you select the object, its Events will be displayed, just as if you were working with any other Visual Basic object.

Note: The Window object events are only valid when a report is sent to a preview window using the Preview method.

The following code demonstrates how to set up and use events for both the Report object and the Window object. Actual event handling code is left for you to fill in. You are limited only by the restrictions of the Visual Basic language.

```
Option Explicit
Public WithEvents rpt1 As CRPEAuto.Report
Public vw1 As CRPEAuto.View
Public WithEvents wnd1 As CRPEAuto.Window
Private Sub Command1_Click()
    Set vw1 = rpt1.Preview
    Set wnd1 = vw1.Parent
End Sub
Private Sub Form_Load()
    Set app1 = CreateObject("Crystal.CRPE.Application")
    Set rpt1 = app1.OpenReport("c:\crw\rt01.rpt")
    rpt1.EventInfo.ActivatePrintWindowEventEnabled = True
    rpt1.EventInfo.ClosePrintWindowEventEnabled = True
    rpt1.EventInfo.GroupEventEnabled = True
    rpt1.EventInfo.PrintWindowButtonEventEnabled = True
    rpt1.EventInfo.ReadingRecordEventEnabled = True
    rpt1.EventInfo.StartStopEventEnabled = True
End Sub
Private Sub rpt1_Start(ByVal Destination As _
                CRPEAuto.CRPrintingDestination, _
                useDefault As Boolean)
    ' Put event handling code here.
End Sub
Private Sub rpt1_Stop (ByVal Destination As
                CRPEAuto.CRPrintingDestination,
                ByVal Status As CRPEAuto.CRPrintingProgress)
    ' Put event handling code here.
End Sub
Private Sub wnd1_ActivatePrintWindow()
    ' Put event handling code here.
End Sub
Private Sub wnd1_ClosePrintWindow (useDefault As Boolean)
    ' Put event handling code here.
End Sub
' Other events for the Report and Window objects
' can be seen by using the Visual Basic Object Browser
' with the Crystal Report Engine Object Library.
' (a lightning bolt icon appears next to Events in
' the Object Browser.)
' Once and instance of a Report object or Window object,
' is declared, you can add Event handlers to your code by
' selecting the object in the Visual Basic Object list and
' then selecting the desired event.
```

For complete descriptions of all available Crystal Report Engine Object Library Events, refer to the `Report Object` and the `Window Object` in the *Techrefvol2.pdf*.

Note: In the previous version of Crystal Reports a report or window object variable declared WithEvents could only be Set once. A VB error occurred if you tried to Set the variable to a different value(i.e. access a new report or display a new Preview window). This problem no longer exists. You can now reset the values of WithEvent object variables.

Distributing the Automation Server with Visual Basic Applications

When you finish designing your application and decide to distribute it to your users, you must make sure that the Crystal Report Engine Automation Server is distributed with it. In addition, you must make sure the automation server gets registered on your users' systems. The easiest way to do this is to use the Application Setup Wizard installed with Visual Basic.

This Wizard leads you through the process of designing a setup application for your project. In addition, the Setup Wizard detects any ActiveX components included in your project and leads you through the process of adding code to the setup application to include the required files and register the components on a user's machine.

For more information about files that need to be distributed with Crystal Report Engine applications, refer to *License Manager Help (License.hlp)*.

Sample Applications

Crystal Reports includes a complete sample application written in Visual Basic 5.0 using the Crystal Report Engine Automation Server. The Xtreme Mountain Bike Inventory Application is a complete real-world application that provides various reports to employees at a fictitious company. Report access is restricted based on user logon information. The application is located in \Program Files\Seagate Software\Crystal Reports\sample\Xtreme\Inventory and provides the option of viewing the source code for any Visual Basic form displayed.

In addition, a self-extracting executable located in the \Program Files\Seagate Software\Crystal Reports\sample\sam32aut (or sam16aut) directory contains three small sample applications that demonstrate various aspects of the Crystal Report Engine Automation Server. Simply run the SAM32AUT.EXE application to install the samples. The three samples are:

- **AUBASIC**
  Demonstrates the basic code required to open a report and print, preview, or export it using the Crystal Report Engine Automation Server.
- **AUBROWSE**
  Demonstrates how to browse through the areas of a report and access the objects in each area.

- **AUFMLA**
  Demonstrates how to get and set record selection formulas, group selection formulas, and SQL queries stored with a report.

Grid Controls and the Crystal Report Engine

In Crystal Reports, a Crystal ActiveX Control can be bound directly to a Visual Basic Data Control. Using the Visual Basic Data Control with the Crystal ActiveX Control offers the following benefits:

- Generating reports in Visual Basic programs is made even easier and does not require an existing .RPT file.
- A powerful feature of Visual Basic is ad-hoc queries that are run by executing SQL statements in the RecordSource property of the Data Control. By directly binding a Crystal Custom Control to a Data Control, users can now create reports of dynaset data which are the results of such ad-hoc queries.

The following topics are discussed in this section:

Bound Report Driver and Bound Report Files

When using Crystal Reports to generate reports from database files of a particular file format (for example, Paradox file format), you need to have the appropriate report driver (i.e., PDBPDX.DLL) to retrieve data from the databases. Similarly, when you generate reports by binding to a Visual Basic Data Control, a Bound Report Driver (PDBBND.DLL) is used to retrieve data from the Data Control. Make sure PDBBND.DLL is in your \WINDOWS\SYSTEM directory or search paths, along with other database drivers.

Crystal ActiveX Control Properties

Several properties are added to the Crystal Custom Control in order to support bound reports. These new properties are described below.

Custom

This property allows you to create bound .RPT files at Visual Basic design time and is not available at runtime. After a bound .RPT file is created, programmers can then use Crystal Reports to customize the report layout or even link the bound data to other database tables.

DataSource (Data Control)

This property can be read / write at design time and runtime. This property is ignored if the ReportSource property is 0 (Report files). To generate bound reports, set this property to the Data Control you want to retrieve data from. The Data Control must already be on the form before this property can be set.

BoundReportFooter (Boolean)

This property can be read / write both at design-time and runtime. This property is ignored if the ReportSource property is 0 (Report files). Default is *False* and the bound reports generated will not have page numbers printed. If set to True, page numbers will be printed at the bottom of a page.

BoundReportHeading (string expression)

This property can be read / write both at design time and runtime. This property is ignored if the ReportSource property is 0 (Report files). It specifies the report title at the beginning of a bound report. If it is blank, no report title will be printed.

ReportSource (numeric expression)

This property can be read / write both at design time and runtime. The allowed values are:

0 - Report files

1 - Bound TrueDBGrid Control

3 - All Data Control Fields

The default value is 0 - Report files, and the ReportFileName property must be assigned to an existing report path name (.RPT). This is equivalent to when the new bound report features were not available and all reports were generated from existing .RPT files.

When set to 1 or 3, the ReportFileName property will be ignored and no .RPT file is needed. Reports will be created using data retrieved from Data Control. The reports generated directly from the Data Control are identical to the reports generated from the respective bound .RPT files created using the (Custom) property described above.

Creating a Bound Report using the Crystal ActiveX Control

1. Add the following controls to your Visual Basic form:
2. On the Data Control:
   - Set the DatabaseName property to the name of the database being reported on.
   - Set the RecordSource property (this can be a database table or a SQL query statement).
3. On the Crystal ActiveX Control:
   - Set the DataSource property to the Data Control (for example, Data1).
   - Set the ReportSource to 3 - All Data Control Fields.
4. On the Command Button, add the following code for the Click event:

```
Private Sub Command1_Click()
    CrystalReport1.Action = 1
End Sub
```

Run the application, click the command button, and the Crystal ActiveX Control will retrieve data from the Data Control and create the report. The report will appear as a simple columnar report. There are no runtime properties to control any report formatting. However, this can be accomplished at design-time by editing the report designed by the ActiveX control (a report template) in Crystal Reports.

Creating a Formatted Bound Report

1. Add the Data control, ActiveX control, and a command button to your form.
2. On the Data control, set the DatabaseName property and the RecordSource property as you did in the previous example.
3. On the ActiveX control:
   - Set the DataSource property to the Data Control (i.e., Data1).
   - Set the ReportSource property to 3 - All Data Control Fields.
   - Open the Custom property and select the Data-Bound Report Tab.
   - Click the Save Report As button and enter a name for the report.
4. Open the report template in Crystal Reports and apply any formatting that you want including spacing between columns, font size, colors, grouping, and totaling. Save the report template again when finished.

5. In your Visual Basic application, set the following properties for the ActiveX control:
   - Set the ReportSource to 0 - Report File.
   - Set the ReportFileName to the .RPT file that you saved (include the complete path of the file).
6. On the command button, add the following code to the Click event:

```
Private Sub Command1_Click()
    CrystalReport1.Action = 1
End Sub
```

Now, the application will create the report at runtime with the formatting you have applied.

Creating a Formatted Bound Report at Runtime

The following steps describe an alternative method of creating formatted bound reports:

1. Create your Visual Basic application as in the first example above.
2. Set the ActiveX Control to print to a preview window, and run the application.
3. Click the **Export** button in the preview window, and export the report to a disk file in .RPT format.
4. Once the report has been exported, you can open it up in Crystal Reports.
5. Perform all formatting changes that you want and save the report.
6. Return to the Visual Basic application and stop it if it is still running.
7. On the ActiveX Control:
8. Set the ReportSource to 0 - Report File.
9. Set the ReportFileName to the .RPT file that you created.
10. Run the Visual Basic application and you will be able to see your bound report with the formatting changes you've made.

Note:

- When creating formatted reports for use with the bound data control in Visual Basic, you will not be able to refresh the data from within Crystal Reports since the data does not exist outside of the Visual Basic application.
- If you plan on using a formatted bound report, you will not be able to modify anything in the SELECT statement of the data control. The report needs all these fields and will fail if they are not all there. The formatted report cannot report on any new fields.

When passing properties at runtime using bound reports (i.e., SortFields), the syntax is slightly different. For example, the following syntax would be used for the Formulas and SortFields properties in a normal report:

```
CrystalReport1.Formulas(0) = "COMMISSION= {TableName.FIELDNAME}"
CrystalReport1.SortFields(0) = "+{TableName.FIELDNAME}"
```

However, for a bound report, the following syntax would be used:

```
CrystalReport1.Formulas(0) = "COMMISSION= {Bound Control.FIELDNAME}"
CrystalReport1.SortFields(0) = "+{Bound Control.FIELDNAME}"
```

Sample Application

Crystal Reports includes a complete sample application written in Visual Basic 5.0 using the Crystal Report Engine Automation Server and the Microsoft Data Bound Grid control. The Xtreme Mountain Bike Inventory Application is a complete real-world application that provides various reports to employees at a fictitious company. The Microsoft Data Bound Grid control is used for an order-entry page that dynamic reports are produced from. The application is installed, by default, in the `\Program Files\Seagate Software\Crystal Reports\sample\Xtreme\Invntory` directory.

Crystal Report Engine 2

The Crystal Report Engine is used alone and as a base class for other Crystal Reports development tools, including the Crystal ActiveX Control, Crystal Report Engine Automations Server, and the Crystal Visual Component Library. In this chapter you will find a general overview of how the Crystal Report Engine is used in an application, as well as a detailed section on the Crystal Report Engine API and it's most common functionality.

Introduction to the Crystal Report Engine

Crystal Reports is a powerful stand-alone report creation application, as well it provides a report writing module that you can add to your own applications. As a developer using C, C++, Visual Basic, ObjectVision, Turbo Pascal, Visual dBASE, Delphi, or any programming language that can access a DLL, you can add sophisticated report generating and printing capabilities to your applications without the time-consuming task of writing your own code.

The Crystal Report Engine is a Dynamic Link Library (DLL) that allows your applications to access the same powerful report printing features that are available in Crystal Reports. As a licensed user of Crystal Reports, you receive royalty-free rights to ship the Crystal Report Engine DLL (CRPE.DLL or CRPE32.DLL) and all of its support files with any application you create.

Note: For more information regarding current runtime file requirements, see the *Crystal Reports Developer Runtime Help (Runtime.hlp)*.

From your application, you can access the Crystal Report Engine through any of several Crystal Report Engine development tools:

- "Crystal ActiveX Controls" on page 10 (CRYSTL32.OCX)
- "Crystal Report Engine Automation Server" on page 12 (CPEAUT.DLL or CPEAUT32.DLL)
- *Crystal Visual Component Library in the Techrefvol2.pdf* (UCRPE.DCU or UCRPE32.DCU)
- *The Crystal Report Engine Class Library in the Techrefvol2.pdf* (PEPLUS.H and PEPLUS.CPP)
- *The Crystal NewEra Class Library in the Techrefvol2.pdf*
- "Crystal Report Engine API" on page 31 (CRPE32.DLL)

When your application runs, it links with the Crystal Report Engine to access report writing functionality. Reporting can be simple, producing only a single report that is sent to a printer or preview window with no options available to the user, or it can be complex, allowing the user to change such things as record selection, sorting, grouping, or export options.

Sample Applications

Crystal Reports comes with a number of sample applications that show you how to incorporate the capabilities of the Crystal Report Engine. Use these applications to further your understanding of the Crystal Report Engine and how to use it in various programming environments.

SQL and ODBC

The Crystal Report Engine is fully compatible with most popular SQL DBMS applications, including Sybase SQL Server, Oracle, Gupta SQLBase, and Microsoft SQL Server. The Crystal Report Engine includes options for logging on to and off of SQL servers and ODBC data sources and also includes the ability to edit the SQL statement passed through to an SQL or ODBC database.

Exporting Reports

The Crystal Report Engine enables you to print to a printer or a preview window with simple function calls. In addition, you can export a file in multiple formats and to multiple destinations. For example:

- through e-mail to another person or group of people
- directly to disk
- to HTML for updating a web site
- to a Microsoft Exchange folder
- to a Lotus Notes folder
- to an ODBC data source

The report can be exported in any of several word processing, spreadsheet, database file, or data exchange formats including HTML.

Before using the Crystal Report Engine in your application

Before you add the Crystal Report Engine to your application, you should be familiar with some key features of the Crystal Report Engine. Review the following points, and make sure you understand each before attempting to make calls to the Crystal Report Engine from your application.

- The Crystal Report Engine outputs existing reports. You cannot create report files using the functionality of the Crystal Report Engine. Reports must be created using the Crystal Reports application described in the Crystal Reports User's Guide. Make sure you understand the report creation process before trying to print reports with the Crystal Report Engine.

Note: Visual Basic programmers can use the Active Data Driver, along with the Crystal Report Engine API or the Crystal Report Engine Automation Server to create reports dynamically at runtime. For more information, refer to "Active Data" on page 563.

- The Crystal Report Engine provides a convenient add-on to your existing application development project. With just a few lines of code, you can produce a powerful report writing and distribution tool that would take thousands of lines of code and weeks to produce otherwise.

- The Crystal Report Engine does not require the use of a fixed user interface. The Crystal Report Engine is designed to work with your existing development project and allows you to define the user interface your customers and users are familiar with and expect from your application.

Using the Crystal Report Engine

Any development project that incorporates the Crystal Report Engine requires three steps:

- "Step 1: Creating reports" on page 28 (The reports that your users access.)
- "Step 2: Designing the user interface that drives the Crystal Report Engine" on page 29.
- "Step 3: Adding the Crystal Report Engine to your application" on page 29.

Related topics:

"Using the Crystal Report Engine API in Delphi" on page 30

Step 1: Creating reports

Creating reports to include with your applications is identical to creating reports for your own use; there are no restrictions. Using the procedures outlined in the *Crystal Reports User's Guide* and *Crystal Reports Online Help (crw.chm)*, create as many kinds of reports as you want to make available to your users. You can make the reports as simple or as sophisticated as your needs dictate.

While designing reports, though, keep in mind their ultimate destination. Some export formats do not support all of the formatting options available in Crystal Reports. For example, if you will be exporting reports to HTML to automatically update a web site, HTML may not support all of the fonts available on your system. This is not a limit of the Crystal Report Engine export functionality, but a limit of the HTML format itself.

If you are a Visual Basic programmer or you are using any development environment that supports Automation Servers, you may want to have reports dynamically designed for you at runtime using the Active data driver. For complete information on using the Active data driver, see "Active Data" on page 563.

Visual Basic programmers can also take advantage of the Visual Basic data control or the TrueGrid ActiveX control at runtime to dynamically produce report files. See "Grid Controls and the Crystal Report Engine" on page 20, for information on using these controls with the Crystal Report Engine.

Step 2: Designing the user interface that drives the Crystal Report Engine

The interface you develop to allow users to print reports is limited only by your needs and your imagination. The kind of user interface you select is unimportant to the Crystal Report Engine.

Common methods of using the Crystal Report Engine include a single menu command that produces a single report, a dialog box allowing several options for printing reports, or a completely separate front-end application that is called by your application. All are acceptable techniques, and each has its advantages. How you design your user interface can depend on any or all of the following:

- The purpose of your application.
- The types of reports your application will use.
- The printing options you want to make available with those reports.
- Whether your application will offer only one report or a choice of several reports.

Consider your application and your reporting needs carefully, and design a User Interface that will use the Crystal Report Engine most efficiently.

Step 3: Adding the Crystal Report Engine to your application

Several different Crystal Report Engine development tools can be used to add the Crystal Report Engine to your application:

- "Crystal ActiveX Controls" on page 10
- "Crystal Report Engine Automation Server" on page 12
- *Crystal Visual Component Library in the Techrefvol2.pdf*
- *The Crystal Report Engine Class Library in the Techrefvol2.pdf*
- *The Crystal NewEra Class Library in the Techrefvol2.pdf*
- "Crystal Report Engine API" on page 31

Be aware that you cannot use two or more of these tools in the same application. For example, you cannot create a Visual Basic application that contains the Crystal ActiveX control and also makes calls to the functions in the Crystal Report Engine API. You must choose one tool to implement the Crystal Report Engine in your project and stick with that tool.

When choosing a Crystal Report Engine tool, consider the following:

- What is your development environment?
- What is your programming ability?
- Do you need to implement the entire Crystal Report Engine or just a few features of it?

For example, the Crystal Class Library for NewEra is specifically designed for Informix NewEra. Therefore, if you are programming in Visual Basic, the Crystal Class Library for NewEra is not an option. The Crystal Report Engine Class Library, on the other hand, is based on the Microsoft Foundation Class Library for C++. To use the Crystal Report Engine Class Library, you must be using a C++ development tool, and you must be using the MFC library.

If you are an experienced programmer, you might consider the Crystal Report Engine API or the Crystal Report Engine Class Library. Novice programmers, on the other hand, may want to take advantage of the easy-to-use features of the Crystal ActiveX control, or the Visual Component Library.

The Crystal Report Engine API consists of a large number of functions exposed directly from the Crystal Report Engine DLL. These functions provide a wide range of power and flexibility for adding report writing features to your own applications.The rest of this chapter discusses the process required to use the Crystal Report Engine API in your own applications.

Although the examples in the following sections concentrate on the C programming language, the concepts should be studied by anyone using the API functions in any language. Additional information specific to Visual Basic programmers using the API can be found in "Enhancements to the Crystal Report Print Engine API" on page 2. Additional information for Delphi programmers is located in "Using the Crystal Report Engine API in Delphi" on page 30. If you wish to use a Crystal Report Engine development tool other than the Crystal Report Engine API, refer to the table of contents for this manual, or search for the name of the programming language or development environment you are using in *Crystal Reports Developer's Help (CrystalDevHelp.chm)*.

Using the Crystal Report Engine API in Delphi

All versions of Delphi can make direct calls to the functions in the Crystal Report Engine API. The Delphi unit file crdelphi.pas includes complete declarations for all Report Engine API functions and records. When you need to add the Report Engine API to your own Delphi unit, simply add the Crystal Report Engine API unit to your project and refer to the unit in your *uses* clause. For example:

```
Uses
    crpe32;
```

The *implementation* section of the Crystal Report Engine API unit contains all of the Crystal Report Engine API functions defined as *external* and as part of the CRPE or CRPE32 DLL.

The "Crystal Report Engine" on page 277, includes Delphi declarations for all Report Engine API functions and records. In addition, the *Crystal Reports Developer's Help (CrystalDevHelp.chm)* includes Delphi sample code using many of the functions and records defined in crdelphi.pas. Search for *Report Engine Functions - Sample Code in Delphi* in the *Crystal Reports Developer's Help (CrystalDevHelp.chm)*.

Crystal Report Engine API

The Crystal Report Engine API (REAPI) is the most direct method of adding the Crystal Report Engine to your application project. The Crystal Report Engine itself is a Dynamic Link Library (DLL), and, therefore, exports its functionality in the form of DLL functions. These functions make up the Crystal Report Engine API.

The Crystal Report Engine DLL, CRPE32.DLL (32-bit), was installed in your \WINDOWS\SYSTEM directory when you installed Crystal Reports. This assures that the DLL is available to any application on your system that uses the Crystal Report Engine.

Note: For complete information on distributing Crystal Report Engine and other runtime DLLs with your application, refer to the *Crystal Reports Developer Runtime Help (Runtime.hlp)*.

The process for loading a DLL and calling DLL functions is a well documented aspect of the Windows API. If you are not familiar with working with DLLs, please refer to Windows API documentation before attempting to use the Crystal Report Engine API. You may also want to consider one of the other methods described in this section for adding the Crystal Report Engine to your application.

The rest of this section assumes an understanding of DLLs and how to use them in a Windows application. It also assumes a basic understanding of the C language. The examples here are written in C, and do not cover the LoadLibrary, GetProcAddress, or FreeLibrary calls.

Many Windows development environments support direct calls to DLL functions, Visual Basic, Visual dBASE, and Delphi, for example. Refer to the documentation for your development environment for complete instructions on using a DLL. Your documentation may also cover instructions on how to translate C function calls to the language you use. Study your documentation, then review the steps described here for using the Crystal Report Engine in an application via the Crystal REAPI.

Declarations for the Crystal Report Engine API (REAPI)

Crystal Reports provides several source code files that declare the functions in the Crystal REAPI for several popular development languages. These files were installed in the Crystal Reports directory (\CRW by default) and are ready to be immediately added to your project. The following Crystal REAPI declaration files are available:

- CRPE.H declares all Crystal Report Engine API functions for C/C++.
- GLOBAL.BAS and GLOBAL32.BAS declare all Crystal Report Engine API functions for Visual Basic. For more information on using the Crystal Report Engine API with Visual Basic, see “Enhancements to the Crystal Report Print Engine API” on page 2.
- CRPEDB.H declares several Crystal Report Engine functions for Visual dBASE. Because of limits in the dBASE language, not all Crystal Report Engine

functions are available to dBASE programmers. Refer to the individual function in *Crystal Reports Developer's Help (CrystalDevHelp.chm)* for information on dBASE availability.

- CRDELPHI.PAS and CRPE32.PAS declare all Crystal Report Engine API functions for Delphi. For more information on using the Crystal Report Engine API with Delphi, see "Using the Crystal Report Engine API in Delphi" on page 30.

Note: Functions can be declared on an individual basis, but unless you will only be using a few of the Crystal Report Engine functions in your code, it is easiest to simply copy one of the previously mentioned code files into your project directory and add it to your project.

Using the Crystal Report Engine API

The Crystal REAPI provides two options for processing and producing reports from within an application:

- "The Print-Only Link" on page 32
- "The Custom-Print Link" on page 36

The Print-Only Link is the fastest, easiest method for producing a report with the Crystal REAPI. A Print-Only Link, however, provides a very limited functionality. It allows a report to be printed on a default printer or previewed in a window on-screen. It does not allow you to customize a report in any way before printing it, though.

A Custom-Print Link, on the other hand, provides all the report processing power of Crystal Reports itself. By coding a Custom-Print Link into your application, you can change record selection, record sorting, group creation, group selecting, group sorting, exporting to disk files, e-mail, Exchange and Lotus Notes folders, ODBC data sources, selecting specific printers for printing, logging on to SQL servers and ODBC data sources, editing formulas, formatting report sections, and much more. A Custom-Print Link is, however, a more complex process to code than a Print-Only Link.

The first time you use the Crystal REAPI in your application project, you may want to start by coding a simple Print-Only Link to produce basic reporting functionality. As your project develops and you become more familiar with the Crystal REAPI, you can expand the reporting functionality with a Custom-Print Link.

The Print-Only Link

A Print-Only Link is performed using the PEPrintReport function. The PEPrintReport function provides basic report printing functionality and demonstrates basic techniques for calling Crystal Report Engine functions from your application.

PEPrintReport enables your application to print a report, to select the output device, either a default printer or a preview window, and to specify the size and location of the preview window if the report is printed to a window. This function does not enable

you to customize the report (select the records to print, set the sort order, etc.) at print time. You can set those parameters at report design time (using the Crystal Reports Design Tab), but you cannot change them at print time through a Print-Only Link.

If the report is sent to a preview window, you should also use the PEOpenEngine and PECloseEngine functions with your Print-Only Link. PEOpenEngine and PECloseEngine allow you to control how long the preview window remains open. The window will remain open until the PECloseEngine function is called or the user clicks Close in the window. If PEOpenEngine and PECloseEngine are not used, and the report is sent to a preview window, the window will automatically close as soon as the report finishes processing.

Note: You may also want to get in the habit of using PEOpenEngine and PECloseEngine in all Print-Only Links, as they are required steps to coding a Custom-Print Link. If your code includes these functions when you design a Print-Only Link, advancing the application to use a Custom-Print Link in the future will be much easier.

PEPrintReport Arguments

PEPrintReport is declared in CRPE.H as follows:

```
short FAR PASCAL PEPrintReport (
        char FAR *reportFilePath,
        BOOL toDefaultPrinter,
        BOOL toWindow, char FAR *title,
        int left, int top,
        int width, int height,
        DWORD style, HWND parentWindow);
```

The following table describes each argument:

Parameter	Description
reportFilePath	The name of the report to be printed. Include the path if the report is not in the current directory. The report name can be hard-coded and unchangeable at runtime, or you can pass a string variable or character array as the result of a user choice.
toDefaultPrinter	If toDefaultPrinter is set to TRUE (1), the report is sent to a printer. The toWindow argument should be set to FALSE.
toWindow	If toWindow is set to TRUE (1), the report is sent to a preview window. The toDefaultPrinter argument should be set to FALSE.
title	The title that you want to appear in the window title bar. This argument can receive a string variable or a character array at runtime.
left	The position, in current screen coordinates, at which you want the left edge of the preview window to appear if the report is being printed to a window. Current screen coordinate measurements can be set within your application.
top	The position, in current screen coordinates, at which you want the top edge of the preview window to appear if the report is being printed to a window. Current screen coordinate measurements can be set within your application.

Parameter	Description
width	The width of your preview window, in current screen coordinates, if the report is being printed to a window. Current screen coordinate measurements can be set within your application.
height	The height of your preview window, in current screen coordinates, if the report is being printed to a window. Current screen coordinate measurements can be set within your application.
style	The style setting, as defined in WINDOWS.H. Style settings can be combined using the bitwise OR operator. These are standard Windows styles. Refer to Windows API documentation for complete information on window styles. Use 0 for default style settings.
parentWindow	Specifies the window handle for the parent window to be used for this preview window.

When designing a Print-Only Link using PEPrintReport, keep the following points in mind:

- If toDefaultPrinter = True, and if you have specified a printer in the report using the Printer Setup command, PEPrintReport prints to the specified printer. Otherwise it prints to the Windows default printer. If you wish to override both the printer specified in the report and the Windows default printer, you will need to establish a Custom-Print Link and specify the printer using the PESelectPrinter function.
- If toDefaultPrinter = True, you may enter null values for all of the remaining parameters except reportFilePath because they apply to printing to a preview window only. The title parameter requires a null string (i.e., “”), while the rest of the parameters will accept 0 (zero).
- If parentWindow is null, Crystal Reports creates a top level window. The top left corner specified is relative to the origin of the screen.
- If parentWindow is the handle of an MDI frame window, Crystal Reports creates a preview window that is an MDI child window with the top left corner relative to the origin of the frame window's client area.
- If parentWindow is the handle of some other window, Crystal Reports creates a preview window that is a child of that window with the top left corner specified relative to the origin of the parent window's client area.
- You can use the Windows constant CW_USEDEFAULT (-32768) as the value of *left*, *top*, *width*, and *height* to indicate a default position for the preview window.

If the preview window is a top-level window and the window style is defined as 0 (i.e., the final two parameters in the PEPrintReport call are 0, 0) or, if the preview window is an MDI child window and the window style is defined as 0, Crystal Reports uses the following default style:

```
(WS_VISIBLE | WS_THICKFRAME | WS_SYSMENU | WS_MAXIMIZEBOX |
WS_MINIMIZEBOX)
```

That is, the default window is a visible window with a thick frame that can be used for sizing the window. The window includes a system menu box, and maximize and minimize buttons.

Example code for a Print-Only Link

The first step in accessing the Crystal Report Engine is to load it into memory. This can be done just before PEPrintReport is called, when a dialog box that allows printing opens, or even when your application first starts.

Once the Crystal Report Engine is open, PEPrintReport can be called as a result of some user action, such as clicking a button on screen, or some internal application procedure.

Finally, once you are finished with the Crystal Report Engine, close it by calling PECloseEngine. If you have several print jobs, do not close the Crystal Report Engine until all print jobs are finished. Opening and closing the Crystal Report Engine uses processor time and should only be performed when necessary.

The following C code demonstrates a possible message handler for an application that provides Print-Only Link functionality through a button in a dialog box. Use this code as an example of how to perform a Print-Only Link.

```
short result;
switch (message)
{
    case WM_INITDIALOG:
        if (!PEOpenEngine())
            ; // Handle error
        return TRUE;
    case WM_DESTROY:
        PECloseEngine();
        return TRUE;
    case WM_COMMAND:
        switch (wParam)
        {
            case IDC_PRINTBUTTON:
                result = PEPrintReport (
                    “boxoffic.rpt”,
                    FALSE, TRUE,
                    “My Report”,
                    CW_USEDEFAULT,
                    CW_USEDEFAULT,
                    CW_USEDEFAULT,
                    CW_USEDEFAULT,
                    CW_USEDEFAULT,
                    hwndParent);
                if (result != 0)
                    return FALSE;
                return TRUE;
        }
        break;
}
```

The Custom-Print Link

A more advanced, and more powerful, method of using the Crystal Report Engine is through a Custom-Print Link. Establishing a Custom-Print Link gives you a great deal of control over your reports at runtime. For example, you can:

- set or modify the report sort order,
- set or modify the record selection and/or group selection formulas,
- modify existing report formulas,
- set or modify the database location,
- capture and evaluate Crystal Report Engine errors,
- export a report to a file, e-mail, Exchange or Lotus Notes folder, or ODBC data source,
- log on to SQL servers and ODBC data sources,
- format report sections,
- and much more.

Note: The Crystal Report Engine allows you to add a selection formula and sort fields to a report at runtime, even if none existed in the report when it was designed. Report formulas created in the Crystal Reports Formula Editor, however, must be added when the report is created in Crystal Reports. A formula can be edited with the Crystal Report Engine, but cannot be added to an existing report from the Crystal Report Engine. Design your reports carefully, and keep this in mind when you create your application.

Coding a Custom-Print Link

There are six required steps to coding a Custom-Print Link in your application. Each uses a different REAPI function. These steps are:

- “Custom-Print Link Step 1: Open the Crystal Report Engine” on page 37 (PEOpenEngine).
- “Custom-Print Link Step 2: Open a print job” on page 37 (PEOpenPrintJob).
- “Custom-Print Link Step 3: Set the output destination” on page 38 (PEOutputToPrinter, PEOutputToWindow, or PEExportTo).
- “Custom-Print Link Step 4: Start the print job” on page 39 (PEStartPrintJob).
- “Custom-Print Link Step 5: Close the print job” on page 39 (PEClosePrintJob).
- “Custom-Print Link Step 6: Close the Crystal Report Engine” on page 39 (PECloseEngine).

In addition to these six steps, you can add several optional tasks any time after Step 2, opening the print job, and before Step 4, starting the print job. These optional tasks include changing selection formulas, editing report formulas, selecting export options, and sorting report fields.

Some REAPI functions can be called at special times to retrieve information about the print job or Crystal Report Engine. For example, PEGetVersion retrieves the current version of the Crystal Report Engine being used and can be called at any time, even without the Crystal Report Engine being open. Another example, PEGetJobStatus, can be called after Step 4 to obtain information about the current status of a job being printed. For more information on all REAPI functions, see "Crystal Report Engine" on page 277 or search for functions by name in *Crystal Reports Developer's Help (CrystalDevHelp.chm)*.

Note: The steps described here apply to a single print job. It is possible to have more than one print job open at once.

Custom-Print Link Step 1: Open the Crystal Report Engine

Example

```
PEOpenEngine();
```

Description

This step starts the Crystal Report Engine and prepares it to accept a print job. The Crystal Report Engine must be open before a print job can be established. You should open the Crystal Report Engine before the user has a chance to try to print a report. For example, if your application uses a dialog box as the user interface to the Crystal Report Engine, open the Crystal Report Engine immediately after the dialog box is created at runtime. Your dialog box can allow the user to establish a print job and make changes to the report while the Crystal Report Engine is already open.

Every time the Crystal Report Engine is opened, it should be closed once your application is finished accessing it ("Custom-Print Link Step 6: Close the Crystal Report Engine" on page 39). For example, if you open the Crystal Report Engine when a dialog box is created, close the Crystal Report Engine when that dialog box is destroyed.

Custom-Print Link Step 2: Open a print job

Example

```
job = PEOpenPrintJob("BOXOFFIC.RPT");
```

Description

When you open a print job, the Crystal Report Engine returns a Job Handle for the print job. This handle is important to identifying the print job in the rest of your code.

To establish a print job, "PEOpenPrintJob" on page 378, requires the path and name of the report that is to be printed. This argument can be hard-coded into the function call, as in the example above, or you can prompt the user to choose a report for printing and pass a variable argument to the function.

To close a print job, refer to “Custom-Print Link Step 5: Close the print job” on page 39. In most cases, you should open the print job immediately before printing and close the print job as soon as the job is finished and the preview window is closed or printing is complete.

Custom-Print Link Step 3: Set the output destination

Example

```
PEOutputToWindow (job, ReportTitle, CW_USEDEFAULT, CW_USEDEFAULT,
CW_USEDEFAULT, CW_USEDEFAULT, 0, NULL);
```

Description

The Crystal Report Engine must know where to send the final report. The report can be printed to a printer, displayed in a preview window, exported to a disk file, exported to another database, or exported to an e-mail address. The example above sends the report to the preview window.

Although you can choose any of the several destinations for report output, you must establish a destination for the report to print. You can, however, write code in your application that allows your users to decide on a destination themselves.

Note: This step does not actually print the report, it only establishes a destination for the report when printed. The report is actually printed in Step 4 using the PEStartPrintJob function.

The following functions are available to establish a print destination:

- “PEOutputToWindow” on page 382
  Printing a report to a window requires no other print destination code other than the function itself.
- “PEOutputToPrinter” on page 380
  Printing a report to a printer requires no other print destination code other than the function itself. However, “PESelectPrinter” on page 389, can be used to select a printer other than the default printer at runtime. The PESelectPrinter function uses the Windows structure “DEVMODE” on page 533. For more information on this structure, refer to the Windows SDK.
- “PEExportTo” on page 299
  The PEExportTo function works with the “PEExportOptions Structure” on page 58 and several DLLs that control a report’s export destination and format. The information required by PEExportTo can be set in your code at design time or it can work with options in your application to allow a user to specify export destination and format. If you would like to allow your users to set the destination and format of a report file, but you do not wish to program the interface to do this, use the “PEGetExportOptions” on page 306 function to have the Crystal Report Engine provide dialog boxes that query the user for export information at runtime.

Custom-Print Link Step 4: Start the print job

Example

```
PEStartPrintJob(job, TRUE);
```

Description

This function actually sends the report to the output device indicated in Step 3. Once "PEStartPrintJob" on page 448, is called, the Crystal Report Engine begins generating the report. The Crystal Report Engine displays a dialog box that indicates the status of the report being generated. If the report is sent to the preview window, the window will appear as soon as PEStartPrintJob is called. The preview window can be closed by a call to "PECloseWindow" on page 290, by closing the Crystal Report Engine (as in Step 6), or by the user clicking the *Close* button.

As a general rule, you should avoid making any formatting changes to a print job once you call PEStartPrintJob, especially if the report is being displayed in a preview window (via PEOutputToWindow). Formatting changes made to a report while it is still being generated and displayed in the preview window may produce undesired results, and can cause errors in the Crystal Report Engine.

Custom-Print Link Step 5: Close the print job

Example

```
PEClosePrintJob(job);
```

Description

Once the print job has completed, it can be closed using "PEClosePrintJob" on page 288. If you wish to make more changes to the report and print it again, you can do so before closing the job. However, once your application is finished with a report, it should close the print job to free up memory in the user's system.

Custom-Print Link Step 6: Close the Crystal Report Engine

Example

```
PECloseEngine();
```

Description

This function closes the Crystal Report Engine entirely. No other Crystal Report Engine functions relating to print jobs may be called once the Crystal Report Engine is closed. Therefore, you should keep the Crystal Report Engine open until it is no longer needed in your application. For example, if the Crystal Report Engine is accessed through a dialog box in your application, you should wait to close the Crystal Report Engine until the dialog box is exited and destroyed by Windows.

A Sample Custom-Print Link

The sample code below has been designed to demonstrate four of the six basic steps in establishing a Custom-Print Link using the C programming language. This example is based on the following scenario:

- Using Crystal Reports, you have created a report called ORDER.RPT and saved it to the C:\CRW directory. This report is a listing of customer orders, and it is the only report your application will need to print.
- In your application, you have created a Print Report menu command that opens a dialog box. The dialog box allows the user to select whether the report is printed to the printer or sent to a preview window. If the report is to be sent to the preview window, a Boolean variable called *ToWindow*, declared and initialized in another section of code not seen here, is given the value of TRUE. If the report is to just be sent straight to the printer, *ToWindow* is given the value FALSE.
- In the Print Report dialog box, there is also a *Print* button that initializes the event procedure to generate and print the report. The *Event code* section below demonstrates how the Custom-Print Link can be coded in the *Print* button event procedure of your application.
- PEOpenEngine is called when the dialog box is created, and PECloseEngine is called when the dialog box is destroyed. For this reason, these two steps are not included in the Custom-Print Link that appears below.

The topic titled Event code demonstrates the basic custom-print event procedure. This code includes *If* statements that check to see if an error has occurred during the call to the Crystal Report Engine. When an error occurs, you can easily handle the error in a separate routine or function. The event code below calls the function ReportError whenever an error occurs. ReportError is not a Crystal Report Engine function but is meant simply as an example of how to handle Crystal Report Engine errors. The code for ReportError appears in the section *Error code*.

Event code

```
short hJob; /* print job handle */
BOOL bResult;
hJob = PEOpenPrintJob("C:\\CRW\\ORDER.RPT");
if (!hJob)
    {
    ReportError(hJob);
    return;
    }
if (ToWindow)
    {
    bResult = PEOutputToWindow(hJob,
            "My Report", CW_USEDEFAULT,
            CW_USEDEFAULT, CW_USEDEFAULT,
            CW_USEDEFAULT, 0, NULL);
    }
```

```
else
    {
    bResult = PEOutputToPrinter(hJob, 1);
    }
if (!bResult)
    {
    ReportError(hJob);
    PEClosePrintJob(hJob);
    return;
    }
if (!PEStartPrintJob(hJob, TRUE))
    {
    ReportError(hJob);
    }
PEClosePrintJob(hJob);
return;
```

Error code

```
void ReportError(short printJob)
    {
    short   errorCode;
    HANDLE  textHandle;
    short   textLength;
    char    *errorText;
    errorCode = PEGetErrorCode(printJob);
    PEGetErrorText (printJob,
                &textHandle,
                &textLength);
    errorText = (char*)malloc(textLength);
    PEGetHandleString(textHandle,
                errorText,
                textLength);
    MessageBox(hWnd, errorText,
            "Print Job Failed",
            MB_OK | MB_ICONEXCLAMATION);
    return;
    }
```

Code Evaluation

Event code

The following is an evaluation of the sample event code that appears above.

```
short hJob; /* print job handle */
BOOL bResult;
```

This section declares two local variables that are important to the remainder of the code. The variable *hJob* will receive the handle to the print job that results from a PEOpenPrintJob call. This handle is required by most Crystal Report Engine functions. *bResult* will be given a TRUE or FALSE value as the result of several

Crystal Report Engine calls. Any time *bResult* receives a FALSE value, an error has occurred.

```
hJob = PEOpenPrintJob("C:\\CRW\\ORDER.RPT");
```

This call opens the new print job according to the path and file name of the report that is to be printed. In this example, the report name is hard-coded in the Crystal Report Engine call. A user would have no choice as to which report is printed. This function could also accept a character array or a pointer to a character array as an argument, allowing you to give your users the opportunity to choose a specific report for printing. PEOpenPrintJob returns a handle to the new print job, *hJob*. This handle will be used in all of the subsequent Crystal Report Engine calls shown here.

```
if (!hJob)
    {
    ReportError(hJob);
    return;
    }
```

This *if* statement verifies whether a valid print job handle was received in the previous line of code. If PEOpenPrintJob returned a value of 0, the print job is invalid and an error is reported. For more information on processing Crystal Report Engine errors, see the Error code section that appears below.

```
if (ToWindow)
    {
    bResult = PEOutputToWindow(hJob,
            "My Report", CW_USEDEFAULT,
            CW_USEDEFAULT, CW_USEDEFAULT,
            CW_USEDEFAULT, 0, NULL);
    }
else
    {
    bResult = PEOutputToPrinter(hJob, 1);
    }
```

ToWindow acts as a Boolean variable that provides information from the user's decision as to whether this report will be printed to a preview window or to a printer. If *ToWindow* holds a TRUE value, then the user has decided to print the report to a preview window.

The *if else* code determines an output destination for the report based on the user's earlier decision. The PEOutputToWindow function prepares the Crystal Report Engine to create a preview window while PEOutputToPrinter directs the Crystal Report Engine to print the report to the default printer. (The printer used by the Crystal Report Engine can be changed with the PESelectPrinter function.) The variable *bResult* receives a FALSE value if an error occurs in either function call.

```
if (!bResult)
    {
    ReportError(hJob);
    PEClosePrintJob(hJob);
    return;
    }
```

Once the appropriate destination function is called, you must verify its success and report an error if *bResult* is FALSE. ReportError is the error handling routine. It is an internal function designed to process any errors that occur during a print job. The function is passed the current value of the *hJob* handle for use in analyzing errors. Search for *Crystal Report Engine Error Codes* in *Crystal Reports Developer's Help (CrystalDevHelp.chm)* for information on processing errors.

Note: ReportError is not a Crystal Report Engine function, but specific to the code appearing here; it is meant only as an example of how to handle Crystal Report Engine errors.

Since a print job has been opened, you must close it after the error is reported using PEClosePrintJob. See below for more information on this function. Finally, the *if* statement causes a return after the error has been reported, thus ending the print job session.

```
if (!PEStartPrintJob(hJob, TRUE))
    {
    ReportError(hJob);
    }
```

PEStartPrintJob actually sends the print job to the printer or a preview window. If the report is printed to a window, PEStartPrintJob creates and opens the window according to the parameters set in the PEOutputToWindow function. If PEStartPrintJob fails (returns FALSE), an error is reported.

```
PEClosePrintJob(hJob);
```

Once the report has printed, this print job can be closed and another one can be started if needed. If the report has been printed to a preview window, PEClosePrintJob does not close the window. The preview window is closed when the *Close* button is clicked, the PECloseWindow function is called, or the PECloseEngine function is called.

```
return;
```

Now that the print job has finished, the event procedure can return, and the application can wait for the next user event to occur.

Error code

```
void ReportError(short printJob)
    {
```

Crystal Report Engine error processing can be most efficiently handled by a separate internal function, such as the one shown here, that is called during a print job. The Event code that is evaluated above calls the ReportError function whenever a Crystal REAPI function returns an error. The code for the ReportError function appears here as an example of how to access and evaluate Crystal Report Engine errors. The error number returned by PEGetErrorCode can be used to control how your application reacts to different types of Crystal Report Engine errors.

Note: The REAPI functions described here, PEGetErrorCode and PEGetErrorText, are specific to REAPI error handling. For complete descriptions of these functions, see "Crystal Report Engine" on page 277, or search for the functions by name in *Crystal Reports Developer's Help (CrystalDevHelp.chm)*. The function PEGetHandleString is used to retrieve variable length strings generated by different REAPI functions.

```
short   errorCode;
HANDLE  textHandle;
short   textLength;
char    *errorText;
```

Completely processing any Crystal Report Engine error requires at least four variables like those above. While only errorCode will be needed to retrieve the Crystal Report Engine error number, the other three variables will all be needed to retrieve the actual error text.

```
errorCode = PEGetErrorCode(printJob);
```

PEGetErrorCode returns a number associated with the error that has occurred. For a list of these error codes and their meanings, search for *Crystal Report Engine Error Codes* in *Crystal Reports Developer's Help (CrystalDevHelp.chm)* or see "Error Codes" on page 545.

```
PEGetErrorText (printJob,
            &textHandle,
            &textLength);
errorText = (char*)malloc(textLength);
PEGetHandleString(textHandle,
            errorText,
            textLength);
```

The error text must be returned in the form of a handle to a variable length string. The handle is used, along with the PEGetHandleString function to obtain the actual error text and store it in a character array. This is a complicated process, and it should be examined carefully if your code is to work.

```
MessageBox(hWnd, errorText,
        "Print Job Failed",
        MB_OK | MB_ICONEXCLAMATION);
```

Once the error has been obtained, you can display error information to the user. This example simply opens a warning message box to alert the user of the problem. Using the error code and the error text, however, you can control Crystal Report Engine error messages any way that you find appropriate for your application.

```
return;
}
```

Once error processing is finished, you can return to processing the print job. If an error has occurred during the print job, however, then the print job should be terminated immediately after the error is processed. Review the evaluation of the event code above for ideas on how to terminate a print job after an error.

Working with Parameter Values and Ranges

Parameters can contain discrete values, ranges, or both discrete values and ranges together. The following discussion outlines how Crystal Reports handles parameter values and ranges.

Before retrieving a parameter current value or range, Call "PEGetParameterValueInfo" on page 354, to determine what type of value(s) are stored. "PEParameterValueInfo" on page 488, member hasDiscreteValues will contain one of the following three constants.

Constant	Description
PE_DR_HASRANGE	Only ranges are present.
PE_DR_HASDISCRETE	Only discrete values are present.
PE_DR_HASDISCRETEANDRANGE	Both discrete values and ranges are present. See guidelines below.

The functions listed below are used to add and retrieve parameter discrete values and parameter ranges. The sequence of functions that you call in your application will depend on whether discrete values, ranges, or a combination of both are present.

PEXXXParameterCurrentValue(s)	PEXXXParameterCurrentRange(s)
"PEGetNParameterCurrentValues" on page 325	"PEGetNParameterCurrentRanges" on page 325
"PEGetNthParameterCurrentValue" on page 337	"PEGetNthParameterCurrentRange" on page 336
"PEAddParameterCurrentValue" on page 279	"PEAddParameterCurrentRange" on page 278

Use the following guidelines when deciding which sequence of functions to call.

PEParameterValueInfo.hasDiscreteValues = PE_DR_HASRANGE

- The parameter field contains only ranges.
- All values will be treated as ranges.
- Use the PEXXXParameterCurrentRange(s) function calls.

PEParameterValueInfo.hasDiscreteValues = PE_DR_HASDISCRETE

- The parameter field contains only discrete values.
- All values will be treated as discrete values.
- Use the PEXXXParameterCurrentValue(s) function calls.

PEParameterValueInfo.hasDiscreteValues = PE_DR_HASDISCRETEANDRANGE

- The parameter field contains both discrete values and ranges.
- All values will be treated as ranges.
- Use the PEXXXParameterCurrentRange(s) function calls.

- You can also call PEAddParameterCurrentValue to add a discrete value, but the discrete value will be stored internally as a range and you will need to call PEGetNParameterCurrentRanges and then PEGetNthParameterCurrentRange when you want to retrieve it. If you try to retrieve the discrete value using PEGetNParameterCurrentValues, 0 will be returned.

Working with section codes

A report, by default, contains five areas: Report Header, Page Header, Details, Report Footer, and Page Footer. Each of those areas can contain one or more sections. When you add groups, subtotals, or other summaries to your report, the program adds Group Header and Group Footer areas as needed, and each of those areas can contain one or more sections as well. Since one report can have a totally different section configuration from the next, Crystal Reports uses calculated section codes to identify the sections in each report.

In Crystal Report Engine API functions that affect report sections, the *sectionCode* parameter encodes the section type, the group number (if the section is a Group Header or Group Footer section), and the section number (if there are multiple sections in an area) together in a single value.

The Crystal Report Engine API also includes macros for encoding section codes (PE_SECTION_CODE, for use with functions that require a section code) and for decoding section codes (PE_SECTION_TYPE, PE_GROUP_N, and PE_SECTION_N, for use with functions that return a section code). The examples that follow show how the encoding and decoding macros can be used.

Note: You cannot pass the above values directly to a function as section codes. You must use the encoding macro to create a valid section code based on one of the above constants.

Encoding

The PE_SECTION_CODE macro allows you to define a section code to pass as a parameter in Crystal Report Engine functions that require a section code. The syntax for the macro is:

```
PE_SECTION_CODE (sectionType, groupNumber, sectionNumber)
```

The PE_AREA_CODE macro allows you to define a corresponding area code. The following syntax is used:

```
PE_AREA_CODE(sectionType,groupN)
```

sectionType

This indicates the report area or section type that the section is in. For section type, use any of the following constants:

Section Type Constant	Value	Description
PE_SECT_REPORT_HEADER	1	Report Header Section
PE_SECT_PAGE_HEADER	2	Page Header Section
PE_SECT_GROUP_HEADER	3	Group Header Section
PE_SECT_DETAIL	4	Detail Section
PE_SECT_GROUP_FOOTER	5	Group Footer Section
PE_SECT_PAGE_FOOTER	7	Page Footer Section
PE_SECT_REPORT_FOOTER	8	Report Footer Section
PE_ALLSECTIONS	0	All Report Sections

groupNumber

Indicates which group the section is in. If the sectionType value indicated is PE_SECT_GROUP_HEADER or PE_SECT_GROUP_FOOTER, the groupNumber is a zero (0) based index for the group section. If the sectionType value is not one of these group section constants, the groupNumber value should always be zero.

sectionNumber

If the report area has been split into more than one section, sectionNumber indicates which section within the area you are using. This value is a zero (0) based index. In other words, the first section in an area is 0, the next section is 1, etc.

Note: The macro PE_SECTION_CODE calculates and returns the section code number; it does not return an error code.

The following example demonstrates how to obtain a section code using the PE_SECTION_CODE macro. The section code obtained here is for the second section in the Group Header 1 area:

```
code = PE_SECTION_CODE(PE_SECT_GROUP_HEADER, 0, 1);
PESetSectionFormat(job, code, &mySectionOptions);
```

In this case you pass the section type (PE_SECT_GROUP_HEADER), the group number (since this is the first group, use the zero indexed group number 0) and section number (since this is the second section in the Group Header, use the zero indexed section number 1). The program uses the encoding macro and returns a section code which is then passed in the PESetSectionFormat call.

When using PE_ALLSECTIONS in your macro, code can be written in one of two ways:

```
code = PE_SECTION_CODE(PE_ALLSECTIONS, 0, 0);
// the code value returned is 0 - NOT an error code
PESetSectionFormat(job, code, &mySectionOptions);
```

or, you can eliminate using the macro all together:

```
PESetSectionFormat(job, PE_ALLSECTIONS, & mySectionOptions)
```

Note: The maximum number of groups is 25 (possible values of 0 to 24). The maximum number of sections is 40 (possible values of 0 to 39).

Decoding

Some Crystal Report Engine functions return section codes. These values can be decoded using one of three macros:

- PE_SECTION_TYPE (sectionCode)
- PE_GROUP_N (sectionCode)
- PE_SECTION_N (sectionCode)

Each macro accepts an encoded section code as a parameter.

In the following example, you determine the number of sections in the report (using PEGetNSections), obtain the section code for each section (using PEGetSectionCode), and then decode the section code using the PE_SECTION_TYPE, PE_GROUP_N, and PE_SECTION_N macros.

```
numSections = PEGetNSections(job);
for (i = 0;i < numSections;i++)
{
    code = PEGetSectionCode(job, loopSectionN);
    areaType = PE_SECTION_TYPE(code);
    groupN = PE_GROUP_N(code);
    sectionN = PE_SECTION_N(code);

    // Perform section specific code here
}
```

Once you've identified the area, group, and section you want, you can then set the section format using code similar to this:

```
PESetSectionFormat(job, code, &mySectionOptions);
```

Note: Earlier versions of Crystal Reports used different section code constants. Those constants have been remapped to the new section code format so reports created with earlier versions of Crystal Reports can run with applications created with the current version.

Section Map

The following map shows the pattern of section code assignment:

Report Header	
1000	First Section in Report Header Area
1025	Second Section in Report Header Area
1050	Third Section in Report Header Area
1075	Fourth Section in Report Header Area
up to 1975	40th Section in Report Header Area
Page Header	
2000	First Section in Page Header Area
2025	Second Section in Page Header Area
2050	Third Section in Page Header Area
2075	Fourth Section in Page Header Area
up to 2975	40th Section in Page Header Area
GH1	
3000	First Section in First Group Header Area
3025	Second Section in First Group Header Area
3050	Third Section in First Group Header Area
3075	Fourth Section in First Group Header Area
GH2	
3001	First Section in Second Group Header Area
3026	Second Section in Second Group Header Area
3051	Third Section in Second Group Header Area
3076	Fourth Section in Second Group Header Area
Details	
4000	First Section in Details Area
4025	Second Section in Details Area
4050	Third Section in Details Area
4075	Fourth Section in Details Area
GF1	
5000	First Section in First Group Footer Area
5025	Second Section in First Group Footer Area
5050	Third Section in First Group Footer Area
5075	Fourth Section in First Group Footer Area

GF2	
5001	First Section in Second Group Footer Area
5026	Second Section in Second Group Footer Area
5051	Third Section in Second Group Footer Area
5076	Fourth Section in Second Group Footer Area
Page Footer	
7000	First Section in Page Footer Area
7025	Second Section in Page Footer Area
7050	Third Section in Page Footer Area
7075	Fourth Section in Page Footer Area
Report Footer	
8000	First Section in Report Footer Area
8025	Second Section in Report Footer Area
8050	Third Section in Report Footer Area
8075	Fourth Section in Report Footer Area

Section Codes in Visual Basic

The following functions provide Visual Basic equivalents.

Create a section code:

This representation allows up to 25 groups and 40 sections of a given type, although Crystal Reports itself has no such limitations.

```
Function PE_SECTION_CODE(sectionType As Integer, groupN As Integer,
sectionN As Integer) As Integer
    PE_SECTION_CODE = (((sectionType) * 1000) + ((groupN) Mod 25) +
(((sectionN) Mod 40) * 25))
End Function
```

Create an area code:

```
Function PE_AREA_CODE(sectionType As Integer, groupN As Integer) As
Integer
    PE_AREA_CODE = PE_SECTION_CODE(sectionType, groupN, 0)
End Function
```

Decode a group number from a section code:

```
Function PE_GROUP_N(sectionCode As Integer) As Integer
    PE_GROUP_N = ((sectionCode) Mod 25)
End Function
```

Decode a section number from a section code:

```
Function PE_SECTION_N(sectionCode) As Integer
   PE_SECTION_N = (((sectionCode \ 25) Mod 40))
End Function
```

Decode a section type from a section code:

```
Function PE_SECTION_TYPE(sectionCode As Integer) As Integer
    PE_SECTION_TYPE = ((sectionCode) \ 1000)
End Function
```

Crystal Report Engine API variable length strings

Several REAPI functions provide information in the form of a variable length string value or character array. When your program calls an REAPI function that produces a variable-length string, the Crystal Report Engine saves the string, creates a string handle which refers to the string, and returns that handle along with a value indicating the length of the string. To retrieve the contents of the string, you must call "PEGetHandleString" on page 318. This approach allows you to allocate a buffer of the exact size needed to hold the string before obtaining the actual string.

If your development language cannot allocate a buffer at runtime, you should declare a reasonably large buffer. Field names and error messages will generally be less than 100 bytes, but formulas may be 1000 bytes or longer. You can control how much data is copied to the buffer when you call PEGetHandleString.

Here is the procedure to follow when obtaining a variable length string:

1. Call-up the function which produces the string. This returns the string handle and length. The length includes all characters in the string plus a terminating null byte.
2. If necessary, allocate the string buffer.
3. Call-up *PEGetHandleString* to copy the string from the handle into the buffer.

Note: PEGetHandleString frees the memory occupied by the string handle, so you can only call this function once for a given handle.

Note: For experienced Windows programmers: text and name handles are Global Memory Handles for memory segments on the global heap. If you prefer, you can access these segments using the Windows GlobalLock, GlobalUnlock, and GlobalFree functions. Contents of name and text handles are null terminated ASCII strings. You must free the text handle with GlobalFree when you are done with it (PEGetHandleString does this for you, if you use it).

Sample Code

Use the following C code as an example of how to call a function that returns a variable length string. The code uses the “var reportAlertInfo : PEReportAlertInfo) : boolean stdcall;” on page 343, function which obtains the name of a field being used to sort the report and the direction of the sort. There are several other functions that return variable length strings, all of which are handled in a similar fashion.

Examine this code carefully and try to incorporate it into your own application without modifying the basic procedure. Only experienced programmers should try making changes to this technique since small mistakes here can cause major errors in your application. If you expect to use several REAPI functions that return variable length strings, you may want to set this code up in a separate function to avoid repetition and errors.

```
HANDLE  nameHandle;
short   nameLength;
short   direction;
char    *fieldName;
PEGetNthSortField (printJob, sortFieldN,
            &nameHandle, &nameLength,
            &direction);
/* allocate fieldName buffer */
fieldName = (char*)malloc(nameLength);
PEGetHandleString (nameHandle,
                fieldName,
                nameLength);
/*
** fieldName now contains name
** of field and nameHandle is no
** longer valid.
*/
```

Note: If you retrieve a string handle but do not retrieve the string itself (i.e., you do not use PEGetHandleString), you should free up the string memory by calling GlobalFree (nameHandle).

Code Evaluation

```
HANDLE  nameHandle;
short   nameLength;
short   direction;
char    *fieldName;
```

Any time you evaluate a function that returns a variable length string, you will need at least three variables:

- a handle to the string,
- a short integer to hold the length of the string, and
- a character array or pointer to a character array.

The direction variable in this example will hold the sort direction and is specific to "PEGetNthSQLExpression" on page 345.

It is important to note that although the PEGetNthSortField function is defined in the Crystal Report Engine as accepting a pointer to a handle (HANDLE*) and a pointer to a short (short*), nameHandle and nameLength are not defined as pointer variables. Instead, they are defined simply as a HANDLE and a short integer, then passed to PEGetNthSortField with the & operator. This technique automatically initializes the variables with the address of the variable itself. Since the PEGetNthSortField function requires the address in memory to place the information, this is the most convenient method to define and pass the variables.

```
PEGetNthSortField (printJob, sortFieldN,
            &nameHandle, &nameLength,
            &direction);
```

The PEGetNthSortField function places a handle to the sort field name in the nameHandle location and the length of the field name (all characters in the name plus a terminating null byte) in the nameLength location. These values will be used to extract the actual field name.

```
/*allocate fieldName buffer*/
fieldName = (char*)malloc(nameLength);
```

Now that you know the actual length of the field name you are trying to obtain, you can allocate exactly the right amount of memory to store that name. The malloc function does this.

Note: Malloc is defined in the C runtime library stdlib.h.

```
PEGetHandleString (nameHandle,
                fieldName,
                nameLength);
```

"PEGetHandleString" on page 318, uses the string handle to retrieve the field name and store it in *fieldName*. At the same time, *nameHandle* is invalidated. Now, the text can be used like any other character string.

Note: This code is meant as a basis for your own code. Although these elements shown here are necessary for extracting a variable length string from certain Crystal Report Engine functions, experienced programmers may wish to expand the code to trap errors or handle the string text differently.

The following is a list of the Crystal REAPI functions that return variable length strings:

- "PEGetAreaFormatFormula" on page 302
- "PEGetErrorText" on page 305
- "PEGetFormula" on page 307
- "PEGetGroupOptions" on page 316
- "PEGetGroupSelectionFormula" on page 317
- "PEGetNthFormula" on page 334
- "PEGetNthGroupSortField" on page 335
- "PEGetNthParameterField" on page 340
- "var reportAlertInfo : PEReportAlertInfo) : boolean stdcall;" on page 343
- "PEGetReportTitle" on page 358
- "PEGetSectionFormatFormula" on page 361
- "PEGetSelectedPrinter" on page 363
- "PEGetSelectionFormula" on page 365
- "PEGetSQLQuery" on page 366

Crystal Report Engine API structures

Several REAPI functions require a structure or user-defined variable type to be passed as one or more arguments. Some of these functions require that you assign values to all members of the structure before calling the function so that the information can be used to make various settings in the Crystal Report Engine. Other functions require only the size of the structure be assigned to the StructSize member. These functions fill in the rest of the structure members for you, providing you with valuable information about a print job.

Note: The term structure is used here to mean both C structures and other user-defined types or records in languages such as Visual Basic and Delphi. If you are unfamiliar with this type of data, refer to the documentation for the programming language you are using.

Each structure used by REAPI is defined and explained in *Crystal Reports Developer's Help (CrystalDevHelp.chm)* with a link to the function that uses it. Functions that use structures also have hypertext links to the structure definitions.

Some of the structures, "PEMouseClickEventInfo" on page 481, for example, are complex, requiring other structures be passed as member values. Not all programming languages support this feature. If you are using a programming language that does not allow the use of a structure variable as a member variable defined inside other structures, declare the member variable as another data type, such as an integer or a variant data type, and assign it a value of 0 (zero) at runtime. The Crystal Report Engine will automatically provide default values or will request information from the user.

Note: Structure variables cannot be created using Visual dBASE. Crystal Report Engine functions requiring structures as parameters are not available to dBASE.

Working with subreports

Your application can have much of the same control over subreports that it has over primary reports. The only exceptions are:

- you cannot open or close a print job while a subreport is open, and
- you can only work with report sections that are actually in the subreport.

For example, subreports do not have page header sections like primary reports do, so you cannot do anything with a subreport that requires a page header section.

Most Crystal Report Engine functions require a print job handle as a parameter. When you supply the handle to a primary report, the functions act on the primary report. When you supply the handle to a subreport, the functions act on the subreport. Getting the handle requires a number of steps.

Opening the primary report

You must first open the primary report using the "PEOpenPrintJob" on page 378 function. When you do this, the program returns a handle to the primary report.

Retrieving an interim subreport handle

You must then identify the subreport you want to open, using the "PEGetNSubreportsInSection" on page 331, and "PEGetNthSubreportInSection" on page 346, functions to do this. When you run the PEGetNthSubreportInSection function, the Crystal Report Engine returns an interim, double-word handle to the subreport you specified.

Retrieving the subreport name

Once you have the handle, use the "PEGetSubreportInfo" on page 367 function to retrieve the name of the subreport. When you run this function, the double-word handle is passed as the subreportHandle argument. The program retrieves the subreport name as the name member of the "PESubreportInfo" on page 507) structure.

Opening the subreport and retrieving the job handle

Now that you have the name of the subreport (the name you assigned the subreport when you created it in Crystal Reports), use the “PEOpenSubreport” on page 379 function to open the subreport. When using this function, you pass the name (or pointer to the name, depending on your development tool) as the subreportName argument. The program then opens the specified subreport and returns a job handle.

Running other Crystal Report Engine functions

Once you have the job handle, you can run any of the other Crystal Report Engine functions with the subreport, passing the subreport job handle as the printJob argument.

Changing report formats

When sending reports to a preview window using “PEOutputToWindow” on page 382, you should always avoid making any formatting changes to a print job once you call “PEStartPrintJob” on page 448. If the first page of a report has been displayed in the preview window, and you make formatting changes to the print job, subsequent pages of the report, if requested, may appear formatted differently than the first page. Depending on the changes made, trying to change report formatting after calling PEStartPrintJob can even cause errors in the Crystal Report Engine.

To avoid such formatting problems, you should get in the habit of formatting the report before starting the print job with PEStartPrintJob. Adding a routine to monitor job status using “PEGetJobStatus” on page 319, can also help avoid conflicts. If you need to display the same report with different formatting options, create two separate print jobs, format each separately, and start each separately.

Exporting reports

Using Crystal Reports, you can give your applications the ability to export reports in a number of word processor and spreadsheet formats, and in a variety of popular data interchange formats as well.

The program includes two export functions, “PEExportTo” on page 299, and “PEGetExportOptions” on page 306. PEExportTo can be used by itself or in conjunction with PEGetExportOptions.

- Use PEExportTo by itself if you want your application to export reports in a fixed format to a fixed destination. Use this alternative, for example, if you want to preset the format and destination for a report and have the application export the report according to your specifications in response to user input.
- Use PEExportTo in conjunction with PEGetExportOptions to export reports in the format and destination your user selects from the Export dialog box at Print time.

PEGetExportOptions can only be used in conjunction with PEExportTo.

PEExportTo overview

The “PEExportTo” on page 299 function uses a structure, “PEExportOptions” on page 458, as part of its argument list. This structure passes format and destination data to the function.

When using the PEExportTo function by itself, you hard code the format and destination data into the structure. Then, when you issue a call to “PEStartPrintJob” on page 448, the program exports the report using the format and destination you specified in the code.

- Most of the format and destination data that you need to enter can be taken from the table in the PEExportTo topic.
- To hard code an export file name or e-mail header information, you will have to pass a second structure as an argument to the PEExportOptions structure. This second structure is defined in the *.h file that corresponds with the destination DLL you have selected.

When using the PEExportTo function in conjunction with the PEGetExportOptions function, you run the PEGetExportOptions function first to:

- retrieve the format and destination data that the user specifies in the Export dialog box, and
- pass that data to the PEExportOptions structure (again, part of the PEExportTo argument list).

Then, when you issue a call to “PEEnableEvent” on page 297, the program exports the report using the format and destination specified by the user.

PEExportOptions Structure

```
struct PEExportOptions
{
WORD StructSize;
    // the size of the structure. Initialize to sizeof PEExportOptions
char formatDLLName [PE_DLL_NAME_LEN];
    // Each export format is defined in a DLL. This is the name of the
    // DLL for the format you select. From table in PEExportTo topic.
    // Requires a null-terminated string. Does not need to include
    // drive, path or extension. For example, uxfsepv is an example of
    // a valid formatDLLName.
DWORD formatType;
    // Some DLLs are used for more than one format. Enter the
    // appropriate value from the table under PEExportTo.
void FAR *formatOptions;
    // Some formats offer additional options (see table in the
    // PEExportTo topic). You can set this element to 0. Then, If the
    // DLLs require more information, they will prompt the user
    // for it. To hard code this information, see the note immediately
    // following this structure.
char destinationDLLName [PE_DLL_NAME_LEN];
    // Each export destination is defined in a DLL. This is the name of
    // the DLL for the destination you select. From table in PEExportTo
    // topic. Requires a null-terminated string. Does not need to
    // include drive, path or extension. For example, uxddisk is an
    // example of a valid destination DLLName.
DWORD destinationType;
    // At the present time, each DLL implements only one destination.
    // You must specify a type here, nonetheless, because the DLL may
    // implement more than one destination someday. See the table under
    // PEExportTo for values to enter here.
void FAR *destinationOptions;
    // Some destinations offer additional options (see table in the
    // PEExportTo topic). You can set this element to 0. Then, If the
    // DLLs require more information, they will prompt the user for
    // it. To hard code this information, see the note immediately
    // following this structure.
WORD nFormatOptionsBytes;
    // Set by 'PEGetExportOptions', ignored by 'PEExportTo'. Both
    // functions use the same structure. PEGetExportOptions uses this
    // information in communicating with the application. The

    // application needs to know how many options bytes are being
    // returned because it may need to copy the options. PEExportTo
    // expects a filled in structure and does not need the byte
    // information because it is not going to copy the options. It uses
    // only a subset of the structure that does not include byte
       information.
WORD nDestinationOptionsBytes;
    // Set by 'PEGetExportOptions', ignored by 'PEExportTo'. See
    // comments for nFormatOptionsBytes above.
};
```

Note: You may choose to hard code the data for formatOptions and destinationOptions. You can set the formatOptions and destinationOptions elements to 0 as indicated. If the DLLs require more information than this, however, they will prompt the user to include more information. To hard code this information, you must define and fill in a structure of the appropriate kind. See the header file for the specified DLL for examples. Once the structure is defined, set the formatOptions or destinationOptions element to the address of the structure. Once PEExportTo returns or finishes, deallocate the formatOptions and destinationOptions structures. You should also deallocate the PEExportOptions structure once PEExportTo returns.

Considerations when using the export functions

The export functions are complex function calls. To avoid errors when exporting report files from your application, keep the following things in mind:

- In order to use "PEGetExportOptions" on page 306 and "PEExportOptions" on page 458, you must be using the version of the Crystal Report Engine (CRPE32.DLL) that came with the Professional Edition of Crystal Reports. If you have an earlier version of CRPE32.DLL installed on your machine and its earlier in the path, the program may find it first and not find the export functions. This can happen particularly if you are upgrading to the Professional Edition of Crystal Reports from the version of Crystal Reports that was shipped with Visual Basic Professional Edition. Visual Basic included an earlier version of CRPE32.DLL. Search your disk and delete or rename earlier versions of CRPE32.DLL, or make appropriate adjustments to your path statement.
- Make sure all format DLLs and destination DLLs are located in the same directory as CRPE32.DLL. Once Windows finds CRPE32.DLL, it will expect all of the DLL files to be in the same directory. Format DLLs are all UXF*.DLL files and Destination DLLs are all UXD*.DLL files. As a general rule, it is best to keep all of these files in the \CRW directory or the directory into which you installed Crystal Reports. Also, make certain that the PATH statement in your AUTOEXEC.BAT file includes \CRW.
- The UXF*.H and UXD*.H header files are only necessary when compiling your application. These files should be copied to the same directory as your application's source files.

Handling preview window events

Using the Crystal Report Engine API, you can create a Windows CALLBACK function to handle events that occur in a preview window. For instance, if a user clicks on a button in the toolbar of the preview window, such as the Zoom button or the Next Page button, Windows registers an event for the preview window.

Using the Event functions in the Crystal REAPI, you can add instructions to your own applications to perform specific actions according to events that occur in a preview window. The sample code below demonstrates how to handle preview window events by creating a CALLBACK function for the preview window, then initializing the preview window with that CALLBACK function in your Crystal Report Engine code. The code can handle toolbar button events, Group Tree events, and even drill-down events.

The Crystal Report Engine API Event functions are only valid when a print job is sent to a preview window using PEOutputToWindow.

```
#define  PE_ERR_INVALIDPARAMETERRANGEINFO 672
#include “crpe.h”
#include “Windows.h”
// The EventCallback function is defined as a standard
// Windows CALLBACK procedure. Return TRUE to allow the
// Crystal Report Engine to provide default behavior.
// Return FALSE to prevent default behavior from being carried out.
// The comment TODO indicates where you need to add event
// handling code specific to your application.
#if defined (WIN32)
BOOL CALLBACK EventCallback (short eventID,
                    void *param, void *userData)
#else
BOOL CALLBACK __export EventCallback (short eventID,
                        void *param, void *userData)
#endif
{
    switch(eventID)
    {
        case PE_CLOSE_PRINT_WINDOW_EVENT:
        case PE_PRINT_BUTTON_CLICKED_EVENT:
        case PE_EXPORT_BUTTON_CLICKED_EVENT:
        case PE_FIRST_PAGE_BUTTON_CLICKED_EVENT:
        case PE_PREVIOUS_PAGE_BUTTON_CLICKED_EVENT:
        case PE_NEXT_PAGE_BUTTON_CLICKED_EVENT:
        case PE_LAST_PAGE_BUTTON_CLICKED_EVENT:
        case PE_CANCEL_BUTTON_CLICKED_EVENT:
        case PE_ACTIVATE_PRINT_WINDOW_EVENT:
        case PE_DEACTIVATE_PRINT_WINDOW_EVENT:
        case PE_PRINT_SETUP_BUTTON_CLICKED_EVENT:
        case PE_REFRESH_BUTTON_CLICKED_EVENT:
        {
            PEGeneralPrintWindowEventInfo * eventInfo =
                (PEGeneralPrintWindowEventInfo *) param;
            ASSERT(eventInfo != 0 && eventInfo->StructSize ==
                PE_SIZEOF_GENERAL_PRINT_WINDOW_EVENT_INFO);
            // TODO
        }
            break;
        case PE_ZOOM_LEVEL_CHANGING_EVENT:
        {
            PEZoomLevelChangingEventInfo * eventInfo =
```

```
        (PEZoomLevelChangingEventInfo *) param;
    ASSERT(eventInfo != 0 && eventInfo->StructSize ==
        PE_SIZEOF_ZOOM_LEVEL_CHANGING_EVENT_INFO);
    // TODO
}
    break;
case PE_GROUP_TREE_BUTTON_CLICKED_EVENT:
{
    PEGroupTreeButtonClickedEventInfo * eventInfo =
        (PEGroupTreeButtonClickedEventInfo *)param;
    ASSERT(eventInfo != 0 && eventInfo->StructSize ==
         PE_SIZEOF_GROUP_TREE_BUTTON_CLICKED_EVENT_INFO);
    // TODO
}
    break;
case PE_CLOSE_BUTTON_CLICKED_EVENT:
{
    PECloseButtonClickedEventInfo *eventInfo =
        (PECloseButtonClickedEventInfo *)param;
    ASSERT(eventInfo != 0 && eventInfo->StructSize ==
        PE_SIZEOF_CLOSE_BUTTON_CLICKED_EVENT_INFO);
    // TODO
}
    break;
case PE_SEARCH_BUTTON_CLICKED_EVENT:
{
    PESearchButtonClickedEventInfo *eventInfo =
        (PESearchButtonClickedEventInfo *)param;

    ASSERT(eventInfo != 0 && eventInfo->StructSize ==
        PE_SIZEOF_SEARCH_BUTTON_CLICKED_EVENT_INFO);
    // TODO
}
    break;
case PE_SHOW_GROUP_EVENT:
{
    PEShowGroupEventInfo * eventInfo =
        (PEShowGroupEventInfo *)param;
    ASSERT(eventInfo != 0 && eventInfo->StructSize ==
        PE_SIZEOF_SHOW_GROUP_EVENT_INFO);
    // TODO
}
    break;
case PE_DRILL_ON_GROUP_EVENT:
{
    PEDrillOnGroupEventInfo * eventInfo =
        (PEDrillOnGroupEventInfo *) param;
    ASSERT(eventInfo != 0 && eventInfo->StructSize ==
        PE_SIZEOF_DRILL_ON_GROUP_EVENT_INFO);
    // TODO
}
    break;
```

```
            case PE_DRILL_ON_DETAIL_EVENT:
            {
                PEDrillOnDetailEventInfo * eventInfo =
                    (PEDrillOnDetailEventInfo *) param;
                ASSERT(eventInfo != 0 && eventInfo->StructSize ==
                    PE_SIZEOF_DRILL_ON_DETAIL_EVENT_INFO);
                // TODO
            }
                break;
            case PE_READING_RECORDS_EVENT:
            {
                PEReadingRecordsEventInfo * readingRecordsInfo =
                    (PEReadingRecordsEventInfo *) param;
                ASSERT(readingRecordsInfo != 0 &&
                    readingRecordsInfo->StructSize ==
                    PE_SIZEOF_READING_RECORDS_EVENT_INFO);
                // TODO
            }
                break;
            case PE_START_EVENT:
            {
                PEStartEventInfo * startEventInfo =
                    (PEStartEventInfo *) param;
                ASSERT(startEventInfo != 0 &&
                    startEventInfo->StructSize ==
                    PE_SIZEOF_START_EVENT_INFO);
                // TODO
            }
                break;
            case PE_STOP_EVENT:
            {
                PEStopEventInfo * stopEventInfo =
                    (PEStopEventInfo *) param;
                ASSERT(stopEventInfo != 0 &&
                    stopEventInfo->StructSize ==
                    PE_SIZEOF_STOP_EVENT_INFO);
                // TODO
            }
                break;
            default:
                break;
        }
        return TRUE;
}
// call this function after open a print job
// before call PEStartPrintJob
BOOL initializeEvent(short printJob)
{
    // initialize window options
    // do not have to set window options to get events,
    // however, some of the events are fired only when
    // certain window options are on.
```

```
    PEWindowOptions windowOptions;
    windowOptions.StructSize = PE_SIZEOF_WINDOW_OPTIONS;
    PEGetWindowOptions(printJob, &windowOptions);
    windowOptions.hasGroupTree = TRUE;
    windowOptions.hasSearchButton = TRUE;
    windowOptions.canDrillDown = TRUE;
    if(!PESetWindowOptions(printJob, &windowOptions))
        return FALSE;
    // enable event.
    // by default, events are disabled.
    PEEnableEventInfo eventInfo;
    eventInfo.StructSize = sizeof(PEEnableEventInfo);
    eventInfo.activatePrintWindowEvent = PE_UNCHANGED;
    eventInfo.closePrintWindowEvent = TRUE;
    eventInfo.startStopEvent = TRUE;
    eventInfo.printWindowButtonEvent = PE_UNCHANGED;
    eventInfo.drillEvent = TRUE;
    eventInfo.readingRecordEvent = TRUE;
    if(!PEEnableEvent(printJob, &eventInfo))
        return FALSE;
    // set tracking cursor, gives the user feedback
    // when the cursor is in the detail area
    // (for a drill-down on detail event)
    // use the default cursor behavior in group area.
    PETrackCursorInfo cursorInfo;
    cursorInfo.StructSize = sizeof(PETrackCursorInfo);
    cursorInfo.groupAreaCursor = PE_UNCHANGED;
    cursorInfo.groupAreaFieldCursor = PE_UNCHANGED;
    cursorInfo.detailAreaCursor = PE_TC_CROSS_CURSOR;
    cursorInfo.detailAreaFieldCursor = PE_TC_IBEAM_CURSOR;
    cursorInfo.graphCursor = PE_UNCHANGED;
    if(!PESetTrackCursorInfo(printJob, &cursorInfo))
        return FALSE;
    // set call back function
    if (!PESetEventCallback(printJob, lEventCallback, 0))
        return FALSE;
    return TRUE;
}
```

Distributing Crystal Report Engine applications

Crystal Reports comes with a royalty-free runtime license for any application that uses the Crystal Report Engine through any of the methods described in this chapter. When distributing a Crystal Report Engine application, you must also distribute several runtime files required by the Crystal Report Engine. These files are listed in the *Crystal Reports Developer Runtime Help (Runtime.hlp)*. Be sure to carefully examine this Help file and distribute the appropriate runtime files with your application. All runtime files are included under the runtime license agreement unless otherwise stated.

Additional sources of information

In addition to the information provided in this chapter, you will find a wide-variety of developer topics in *Crystal Reports Developer's Help (CrystalDevHelp.chm)*. Many of these topics contain sample code in C, Visual dBASE, Delphi, and Visual Basic that you can copy directly into your application. For a list of all developer topics, see *Crystal Reports Developer's Help (CrystalDevHelp.chm)*.

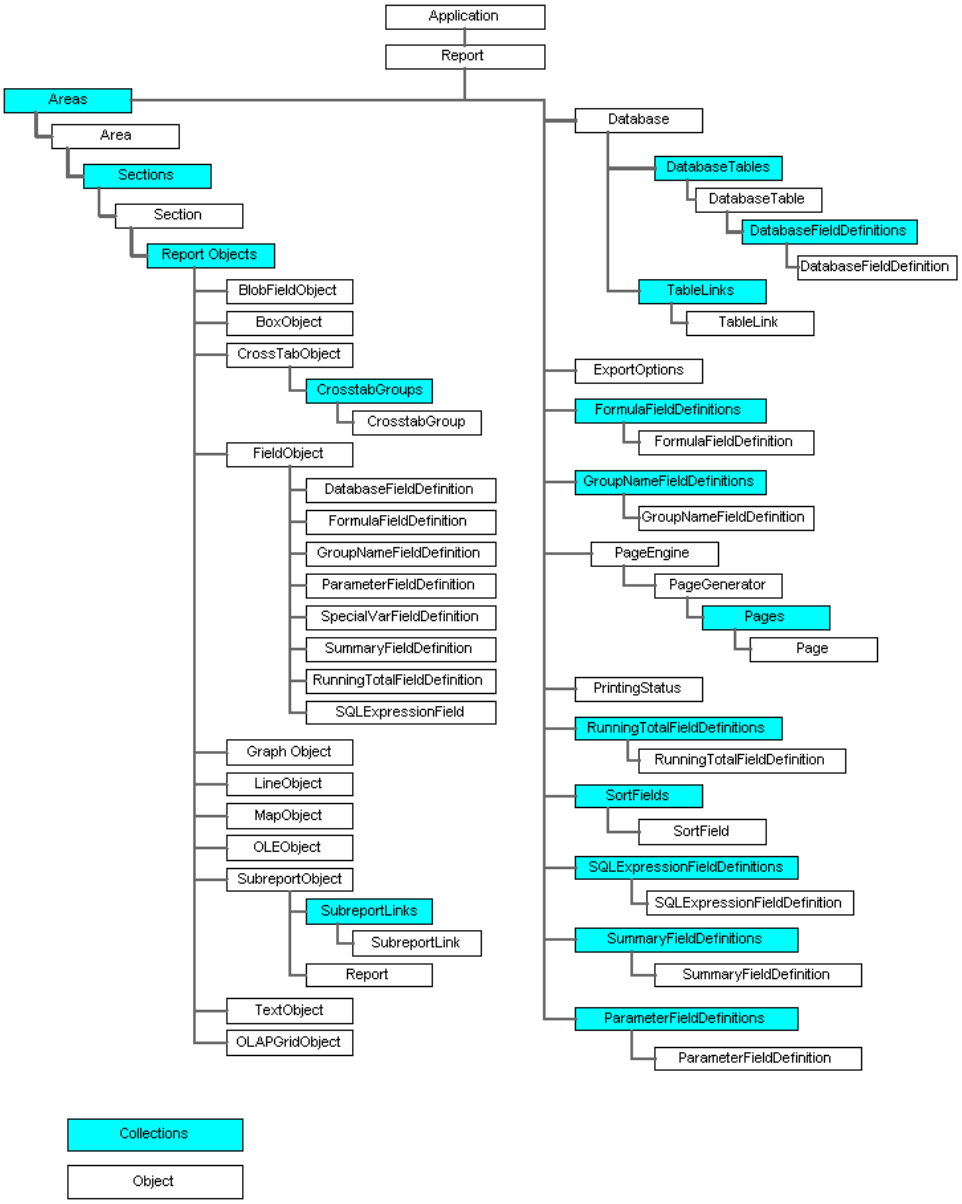
If you are working with the Crystal Report Engine API in Visual Basic, refer to "Enhancements to the Crystal Report Print Engine API" on page 2, for information specific to Visual Basic. Delphi programmers can find information specific to using the Crystal Report Engine API with Delphi under *Crystal Visual Component Library in the Techrefvol2.pdf*.

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 8.5 ActiveX Designer Design and Runtime Library) is a new unified object model that combines the runtime capabilities of the Craxdrt.dll (Crystal Reports 8.5 ActiveX Designer Run Time Library) with the design time capabilities of the Craxddt.dll (Crystal Reports 8.5 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 453 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.

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 Royalty help file (Royalty Required Runtime.hlp).

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.[8]. 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 of the dll.
Dim app As Application
Set app = CreateObject("CrystalRuntime.Application.8")
```

Application Object Methods

The following methods are discussed in this section:

CanClose Method (Application Object)

The CanClose method indicates whether or not the "Application Object" on page 68 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" on page 150 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.

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 hexidecimal.

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)” on page 71. This method is only valid if you have purchased Crystal Reports 6.0 or later.

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” on page 150, 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)” on page 247, 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)" on page 70, or until the "Application Object" on page 68, is destroyed and craxdrt.dll is unloaded. This method is only valid if you have purchased Crystal Reports 6.0 or later.

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 (i.e., "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)" on page 70, or until the "Application Object" on page 68, is destroyed and craxdrt.dll is unloaded. This method is only valid if you have purchased Crystal Reports 6.0 or later.

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” on page 150, 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	Specifies whether you want to open the report exclusively. If you do not provide this parameter the report is opened exclusively and cannot open it a second time.

Returns

- Returns an instance of the "Report Object" on page 150, if the report was successfully opened.
- Returns 0 if the report file does not exist or if an error occurs.

SetMatchLogOnInfo Method (Application Object)

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

Syntax

```
Sub SetMatchLogOnInfo (bl As Boolean)
```

Parameter

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

SetMorePrintEngineErrorMessages Method (Application Object)

Use the SetMorePrintEngineErrorMessages method 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)

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 deatil area height, in twips.	Read only	None
DetailWidth	Long. Gets multiple column report detail area width, in twips.	Read only	None
DiscardOther Groups	Boolean. Gets or sets the discard other groups option.	Read / Write	
Enable Hierarchical GroupSorting	Boolean. Gets or sets group hierarchically flag.	Read / Write	Fromatting idle.
GroupCondition	“CRGroupCondition” on page 216. Gets or sets the group condition.	Read / Write	Can be written only when formatting idle.
GroupCondition Field	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	Fromatting idle.
GroupNumber	Integer. If the area is a group area, this returns the group number. Otherwise, exception is thrown.	Read only	None
HideFor DrillDown	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	“FieldDefinition Object” on page 121. Gets the instance ID field	Read only	Fromatting idle.

Property	Description	Read/Write	Restriction in event handler
KeepGroup Together	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" on page 207. 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.
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.
NumberOfTop OrBottomN Groups	Integer. Gets or sets the number of top or bottom groups.	Read / Write	Formatting idle.
Parent	"Report Object" on page 150. Gets reference to the parent object.	Read Only	None
ParentIDField	"FieldDefinition Object" on page 121. Gets the parent ID field.	Read only	Formatting idle.
PrintAtBottom OfPage	Boolean. Gets or sets the print at bottom of page option.	Read / Write	Can be written only when formatting idle.
RepeatGroup Header	Boolean. Gets or sets the repeating group header option.	Read / Write	Can be written only when formatting idle.
ResetPage NumberAfter	Boolean. Gets or sets the reset page number after option.	Read / Write	Can be written only when formatting idle.
Sections	"Sections Collection" on page 185. Gets a Collection of all the sections in the area.	Read Only	None
SortDirection	"CRSortDirection" on page 228. Gets or sets the group sort direction.	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.
TopOrBottom NGroupSort Order	"CRTopOrBottomNGroupSortOrder" on page 230. Gets or sets the top or bottom N group sort order.	Read / Write	Formatting idle.
TopOrBottom NSortField	"SummaryFieldDefinition Object" on page 198. Gets or sets the top or bottom n sort field.	Read / Write	Formatting idle.

Area Object Methods

The following methods are discussed in this section:

“SetInstanceIDField Method (Area Object)” on page 76

“SetParentIDField (Area Object)” on page 76

SetInstanceIDField Method (Area Object)

Use 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 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” on page 74, 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” on page 74. 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” on page 150. 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

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.
BottomCropping	Long. Gets or sets bottom cropping size, in twips.	Read / Write	Can be written only when formatting idle.
BottomLineStyle	“CRLineStyle” on page 218. Gets or sets the bottom line style.	Read / Write	Can be written only when formatting idle or active.

Property	Description	Read/Write	Restriction in event handler
CloseAtPageBreak	Boolean. Gets or sets the close border on page break.	Read / Write	Can be written only when formatting idle or active.
Field	"DatabaseFieldDefinition Object" on page 92. Gets the database field definition object containing information about the BLOB field.	Read only	None
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.
KeepTogether	Boolean. Gets or sets the keep object together option.	Read / Write	Can be written only when formatting idle or active.
Kind	"CRObjectKind" on page 221. 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" on page 218. 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" on page 174. 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" on page 218. 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" on page 218. Gets or sets the top line style.	Read / Write	Can be written only when formatting idle or active.

Property	Description	Read/Write	Restriction in event handler
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

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” on page 174. 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.
CornerEllipseHeight	Long. Gets or sets the corner elipse height, in twips.	Read / Write	
CornerEllipseWidth	Long. Gets or sets the corner elipse width, in twips.	Read / Write	
ExtendToBottomOf Section	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.

Property	Description	Read/Write	Restriction in event handler
HasDropShadow	Boolean. Gets or sets the border drop shadow option..	Read / Write	Can be written only when formatting idle.
Kind	"CRObjectKind" on page 221. 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	"CRLineStyle" on page 218. 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" on page 174. 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 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.

CrossTabGroup Object

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

CrossTabGroup Properties

Property	Description	Read/Write	Restriction in event handler
BackColor	OLE_COLOR. Gets or sets the crosstab group background color.	Read / Write	Idle

Property	Description	Read/Write	Restriction in event handler
Condition	"CRGroupCondition" on page 216. 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
Field	"FieldDefinition Object" on page 121. Gets or sets the crosstab group's field.	Read / Write	Idle
Parent	"CrossTabObject Object" on page 82. Gets reference to the crosstab group parent object.	Read only	None
SortDirection	"CRSortDirection" on page 228. Gets or sets the crosstab group sort direction.	Read / Write	Idle

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" on page 80. Gets an item from the Collection.	Read only	None
Parent	"CrossTabObject Object" on page 82. 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 82

"Delete Method (CrossTabGroups Collection)" on page 82

Add Method (CrossTabGroups Collection)

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)

Use 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.

Property	Description	Read/Write	Restriction in event handler
BottomLineStyle	"CRLineStyle" on page 218. 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.
ColumnGrandTotal Color	OLE_COLOR. Gets or sets the column grand total color.	Read / Write	Idle
ColumnGroups	"CrossTabGroups Collection" on page 81. Gets the column groups Collection.	Read only	None
EnableKeep ColumnsTogether	Boolean. Gets or sets the enable keep columns together option.	Read / Write	Idle
EnableRepeatRow Labels	Boolean. Gets or sets the enable repeat row labels option.	Read / Write	Idle
EnableShowCell Margins	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
EnableSuppress ColumnGrand Totals	Boolean. Gets or sets the enable suppress column grand totals option.	Read / Write	Idle
EnableSuppress EmptyColumns	Boolean. Gets or sets the enable suppress empty columns option.	Read / Write	Idle
EnableSuppress EmptyRows	Boolean. Gets or sets the enable suppress empty rows option.	Read / Write	Idle
EnableSuppress RowGrandTotals	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.

Property	Description	Read/Write	Restriction in event handler
Kind	"CRObjectKind" on page 221. 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" on page 218. 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" on page 174. Gets reference to the parent object.	Read only	None
RightLineStyle	"CRLineStyle" on page 218. Gets or sets the right line style.	Read / Write	Can be written only when formatting idle or active.
RowGrandTotal Color	OLE_COLOR. Gets or sets the row grand total color.	Read / Write	Idle
RowGroups	"CrossTabGroups Collection" on page 81. Gets the row groups Collection.	Read Only	None
SummaryFields	"ObjectSummaryFieldDefinitions Collection" on page 125. Gets the summary fields.	Read Only	None
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" on page 218. 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 66.

Database Object Properties

Property	Description	Read/Write	Restriction in event handler
Links	“TableLinks Collection” on page 202. Gets database link collection.	Read only	None
Parent	“Report Object” on page 150. Gets reference to the parent object.	Read only	None
Tables	“DatabaseTables Collection” on page 98. 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)

Use 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 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.

ConvertDatabaseDriver Method (Database Object)

Use ConvertDatabaseDriver to convert the database driver DLL used by the report.

Syntax

```
Sub ConvertDatabaseDriver (pDLLName As String, blDoImmediateConvert 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 databse 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 "Re-mapping altered database fields" 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
```

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" on page 150, 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)" on page 247, 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 craxdrt.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)” on page 94 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” on page 229 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
DatabaseFieldName	String. Specifies the name of the field in the database (for example, Product ID).	Read only	None
Kind	"CRFieldKind" on page 212. 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" on page 150. 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.
TableAliasName	String. Gets the alias for the table containing the field.	Read only	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" on page 212. 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" on page 92 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" on page 92, in the collection.	Read only	None
Item (index As Long)	"DatabaseFieldDefinition Object" on page 92. 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" on page 93. 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, DatabaseFieldDefintion(1)).

DatabaseTable Object

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

DatabaseTable Object Properties

Property	Description	Read/Write	Restriction in event handler
ConnectBuffer String	String. Gets the table connect buffer string.	Read only	None
DatabaseType	"CRDatabaseType" on page 209. Gets the database table type	Read only	
DescriptiveName	String. Gets the table descriptive name.	Read only	None
DllName	String. Gets the table driver DLL name.	Read only	None

Property	Description	Read/Write	Restriction in event handler
Fields	"DatabaseFieldDefinitions Collection" on page 93. 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.
LogOnDatabase Name	String. Gets the logon database name.	Read only	None
LogOnServerName	String. Gets the logon server name.	Read only	None
LogOnUserID	String. Gets the logon user ID.	Read only	None
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" on page 85. Reference to the parent object.	Read only	None
SessionUserID	String. Gets the session user ID.	Read only	None
SubLocation	String. Gets the table sublocation.	Read only	None

DatabaseTable Object Methods

The following methods are discussed in this section:

CheckDifferences Method (DatabaseTable Object)

Use 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" on page 229. 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.

SetLogOnInfo Method (DatabaseTable Object)

The SetLogOnInfo method logs on to the data source so table data can be 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.

SetSessionInfo Method (DatabaseTable Object)

The SetSessionInfo method allows the user to log on to a secured Access session.

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)" on page 96.

SetTableLocation Method (DatabaseTable Object)

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

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 (index As Long)	"DatabaseTable Object" on page 93. 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" on page 85. 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 98

"Delete Method (DatabaseTables Collection)" on page 99

Add Method (DatabaseTables Collection)

The Add method is used 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"
```

Delete Method (DatabaseTables Collection)

Use 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" on page 150.

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
CharStringDelimiter	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	"CRExportDestinationType" on page 210. Gets or sets the export destination type.	Read / Write	None
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" on page 207. 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

Property	Description	Read/Write	Restriction in event handler
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" on page 210. Gets or sets the Exchange destination type for reports exported to Exchange folders.	Read / Write	None
ExchangeFolderPath	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
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	"CRExportFormatType" on page 211. 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

Property	Description	Read/Write	Restriction in event handler
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
NumberOfLines PerPage	Integer. Gets or sets the number of lines to appear per page option for reports exported to a paginated text format.	Read / Write	None
ODBC DataSourceName	String. Gets or sets the ODBC data source for reports exported to ODBC.	Read / Write	None
ODBCDataSource Password	String. Sets the ODBC data source password.	Write only	None
ODBCDataSource UserID	String. Gets or sets the user name used to access an ODBC data source for reports exported to ODBC.	Read / Write	None
ODBCExportTable Name	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
PDFExportAllPages	Boolean. Gets or sets wether 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
PDFFirstPage Number	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
PDFLastPage Number	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
Parent	“Report Object” on page 150. Reference to the parent object.	Read only	None

Property	Description	Read/Write	Restriction in event handler
RTFExportAllPages	Boolean. Gets or sets wether 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
RTFFirstPage Number	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
RTFFLastPage Number	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
UseReportDate Format	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
UseReportNumber Format	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
XMLAllowMultiple Files	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 *Crystal Reports Users Guide*.	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 104

"Reset Method (ExportOptions Object)" on page 104

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)	"FieldDefinition Object" on page 121. Gets the collection item.	Read only	None
Parent	IReportObject. Reference to the parent object (ReportObject).	Read only	None

FieldDefinitions Collection Methods

The following methods are discussed in this section:

"Add Method (FieldDefinitions Collection)" on page 105

"Delete Method (FieldDefinitions Collection)" on page 105

Add Method (FieldDefinitions Collection)

Use 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 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.

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
MappingToField Index	Integer. Gets or sets the index of the mapping to field index in the field list of the new database.	Read / Write	None
TableName	String. Gets or sets field's table name.	Read / Write	None
ValueType	"CRFieldValueType" on page 212. Gets or sets the value type that is in the field.	Read / Write	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 168, (for example, ReportObjects.Item(Index), where the index can be the 1-based index number of the item or an Object name).

Property	Description	Read/ Write	Restriction in event handler
AmPmType	"CRAMPMType" on page 207. 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	"CRBooleanOutputType" on page 208. Gets or sets the Boolean output type.	Read / Write	Can be written only when formatting idle or active.

Property	Description	Read/ Write	Restriction in event handler
BorderColor	OLE_COLOR. Gets or sets the object border color.	Read / Write	Can be written only when formatting idle or active.
BottomLineStyle	"CRLineStyle" on page 218. 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.
CurrencyPosition Type	"CRCurrencyPositionType" on page 208. 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" on page 208. Gets or sets the currency symbol type.	Read / Write	Can be written only when formatting idle or active.
DateCalendarType	"CRDateCalendarType" on page 209. Gets or sets the date calendar type.	Read / Write	Can be written only when formatting idle.
DateEraType	"CRDateEraType" on page 209. 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" on page 209. 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.
DateSecond Separator	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.
DateWindows DefaultType	"CRDateWindowsDefaultType" on page 209. Gets or sets the date windows default type.	Read / Write	Can be written only when formatting idle.
DayType	"CRDayType" on page 210. 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
DisplayReverseSign	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, Database FieldDefinition Object, Parameter FieldDefinition Object, etc.).	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" on page 207. 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" on page 216. 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" on page 221. Gets which kind of object (for example, box, cross-tab, field, etc.).	Read only	None
LeadingDayPosition	"CRLeadingDayPosition" on page 217. 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" on page 217. 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.

Property	Description	Read/ Write	Restriction in event handler
LeftLineStyle	"CRLineStyle" on page 218. 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
LineSpacingType	"CRLineSpacingType" on page 218. Gets the line spacing type.	Read Only	None
MaxNumberOf Lines	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.
MinuteSecond Separator	String. Gets or sets minute second separator.	Read / Write	Can be written only when formatting idle or active.
MinuteType	"CRMinuteType" on page 219. Gets or sets the minute type.	Read / Write	Can be written only when formatting idle or active.
MonthType	"CRMonthType" on page 220. 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" on page 220. 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" on page 174. 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" on page 218. Gets or sets the right line style.	Read / Write	Can be written only when formatting idle or active.
RoundingType	"CRRoundingType" on page 226. Gets or sets the number rounding type.	Read / Write	Can be written only when formatting idle or active.
SecondType	"CRSecondType" on page 227. 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.

Property	Description	Read/ Write	Restriction in event handler
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.
TextFormat	"CRTextFormat" on page 230. Gets or sets the text format option for string memo fields.	Read / Write	Can be written only when formatting idle.
TextRotationAngle	"CRRotationAngle" on page 226. 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	"CRTimeBase" on page 230. 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	"CRLineStyle" on page 218. 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.
UseOneSymbol PerPage	Boolean. Gets or sets the use one symbol per page option.	Read / Write	Can be written only when formatting idle or active.
UseSystem Defaults	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" on page 231. 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)

Use 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" on page 218.

SetUnboundFieldSource Method (FieldObject Object)

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

Syntax

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

Parameter

Parameter	Description
pUnboundField Source	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” on page 212. 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.
NumberOfBytes	Integer. Gets the number of bytes required to store the field data in memory.	Read only	None

Property	Description	Read/ Write	Restriction in event handler
Parent	"Report Object" on page 150. 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" on page 212. Gets which type of value is found in the field.	Read only	None

Remarks

Crystal Reports 8.0 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" on page 150. The default syntax is Crystal Report syntax (crCrystalSyntaxFormula).

FormulaFieldDefinition Object Methods

The following methods are discussed in this section:

"Check Method (FormulaFieldDefinition Object)" on page 113

Check Method (FormulaFieldDefinition Object)

The Check method checks formula 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” on page 112, 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” on page 114. Gets collection item. See Remarks below.	Read only	None
Parent	“Report Object” on page 150. 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 114

“Delete Method (FormulaFieldDefinitions Collection)” on page 115

Add Method (FormulaFieldDefinitions Collection)

Use 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 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.

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, etc.).

GraphObject Object Properties

Property	Description	Read/ Write	Restriction in event handler
AutoRange Data2Axis	Boolean. Gets or sets the auto range option for data2 axis. See Remarks below.	Read / Write	Can be written only when formatting idle.
AutoRange DataAxis	Boolean. Gets or sets the auto range option for data axis. See Remarks below.	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” on page 207. 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.
BottomLine Style	“CRLineStyle” on page 218. Gets or sets the bottom line style.	Read / Write	Can be written only when formatting idle or active.
CloseAt PageBreak	Boolean. Gets or sets the close border on page break option.	Read / Write	Can be written only when formatting idle or active.
Condition Fields	“FieldDefinitions Collection” on page 104. Gets the condition fields Collection.	Read only	None

Property	Description	Read/ Write	Restriction in event handler
CrossTab Object	"CrossTabObject Object" on page 82. Gets the crosstab object if this is a CrossTab chart.	Read only	None
Data2Axis Division Method	"CRDivisionMethod" on page 210. Gets or sets the data2 axis division method.	Read / Write	Can be written only when formatting idle.
Data2Axis Division Number	Long. Gets or sets the data2 axis division number.	Read / Write	Can be written only when formatting idle.
Data2Axis Gridline	"CRGridlineType" on page 215. Gets or sets the data2 axis grid line type.	Read / Write	Can be written only when formatting idle.
Data2Axis Number Format	"CRNumberFormat" on page 220. Gets or sets the data2 axis number format.	Read / Write	Can be written only when formatting idle.
DataAxis Division Method	"CRDivisionMethod" on page 210. Gets or sets the data axis division method.	Read / Write	Can be written only when formatting idle.
DataAxis Division Number	Long. Gets or sets the data axis division number.	Read / Write	Can be written only when formatting idle.
DataAxis Gridline	"CRGridlineType" on page 215. Gets or sets the data axis grid line type.	Read / Write	Can be written only when formatting idle.
DataAxis Number Format	"CRNumberFormat" on page 220. Gets or sets the data axis number format.	Read / Write	Can be written only when formatting idle.
DataLabel Font	IFontDisp. Gets or sets standard OLE font for chart data labels.	Read / Write	Can be written only when formatting idle.
DataPoint	"CRGraphDataPoint" on page 213. Gets or sets the graph data points on risers.	Read / Write	Can be written only when formatting idle.
DataTitle Font	IFontDisp. Gets or sets standard OLE font for chart data title.	Read / Write	Can be written only when formatting idle.
DataType	"CRGraphDataType" on page 213. Returns which type of data used in the graph.	Read only	None
DataValue Number Format	"CRNumberFormat" on page 220. Gets or sets the data value number format.	Read / Write	Can be written only when formatting idle.
EnableAuto Scale DataAxis	Boolean. Gets or sets the data axis auto-scale option.	Read / Write	Can be written only when formatting idle.
EnableAuto Scale Data2Axis	Boolean. Gets or sets the data2 axis auto-scale option.	Read / Write	Can be written only when formatting idle.

Property	Description	Read/ Write	Restriction in event handler
EnableFor EachRecord	Boolean. Gets or sets the enable for each record option.	Read / Write	Can be written only when formatting idle.
EnableShow Legend	Boolean. Gets or sets the show legend option.	Read / Write	Can be written only when formatting idle.
Enable Summarize Values	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.
Footnote Font	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.
Graph Direction	"CRGraphDirection" on page 214. Gets or sets the graph direction.	Read / Write	Can be written only when formatting idle.
GraphType	"CRGraphType" on page 214. Gets or sets the graph type.	Read / Write	Can be written only when formatting idle.
GroupAxis Gridline	"CRGridlineType" on page 215. Gets or sets the group axis grid line type.	Read / Write	Can be written only when formatting idle.
GroupLabel Font	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.
GroupTitle Font	IFontDisp. Gets or sets standard OLE font for chart group title.	Read / Write	Can be written only when formatting idle.
HasDropSh adow	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.
IsFootnote ByDefault	Boolean. Gets or sets footnote default flag.	Read / Write	Can be written only when formatting idle or active.
IsGroups TitleBy Default	Boolean. Gets or set groups title default flag.	Read / Write	Can be written only when formatting idle or active.
IsSeriesTitle ByDefault	Boolean. Gets or sets series title default flag.	Read / Write	Can be written only when formatting idle or active.
IsSubTitle ByDefault	Boolean. Get or sets subtitle default flag.	Read / Write	Can be written only when formatting idle or active.
IsTitleBy Default	Boolean. Gets or sets title default flag.	Read / Write	Can be written only when formatting idle or active.

Property	Description	Read/ Write	Restriction in event handler
Keep Together	Boolean. Gets or sets keep object together option.	Read / Write	Can be written only when formatting idle or active.
Kind	"CRObjectKind" on page 221. Gets which kind of object (for example, box, cross-tab, 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.
LeftLine Style	"CRLineStyle" on page 218. 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.
Legend Layout	"CRPieLegendLayout" on page 224. Gets or sets the legend layout for a pie chart.	Read / Write	Can be written only when formatting idle.
Legend Position	"CRLegendPosition" on page 218. Gets or sets the legend position.	Read / Write	Can be written only when formatting idle.
Marker Shape	"CRMarkerShape" on page 219. Gets or sets the marker shape.	Read / Write	Can be written only when formatting idle.
MarkerSize	"CRMarkerSize" on page 219. Gets or sets the marker size.	Read / Write	Can be written only when formatting idle.
MaxData2 AxisValue	Double. Gets or sets data2-axis max value. See Remarks below.	Read / Write	Can be written only when formatting idle.
MaxData AxisValue	Double. Gets or sets data-axis max value. See Remarks below.	Read / Write	Can be written only when formatting idle.
MinData2 AxisValue	Double. Gets or sets data2-axis min value. See Remarks below.	Read / Write	Can be written only when formatting idle.
MinData AxisValue	Double. Gets or sets data-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" on page 174. Gets reference to the parent object.	Read only	None
PieSize	"CRPieSize" on page 224. Gets or sets the size for pie charts.	Read / Write	Can be written only when formatting idle.
RightLine Style	"CRLineStyle" on page 218. Gets or sets the right line style.	Read / Write	Can be written only when formatting idle or active.
SeriesAxis Gridline	"CRGridlineType" on page 215. Gets or sets the series axis grid line type.	Read / Write	Can be written only when formatting idle.
SeriesLabel Font	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.

Property	Description	Read/ Write	Restriction in event handler
SeriesTitle Font	IFontDisp. Gets or sets standard OLE font for chart series title.	Read / Write	Can be written only when formatting idle.
SliceDetach ment	"CRSliceDetachment" on page 227. 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.
SubTitle Font	IFontDisp. Gets or sets standard OLE font for chart subtitle.	Read / Write	Can be written only when formatting idle.
Summary Fields	"SummaryFieldDefinitions Collection" on page 200. 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.
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.
TopLine Style	"CRLineStyle" on page 218. Gets or sets the top line style.	Read / Write	Can be written only when formatting idle or active.
Viewing Angle	"CRViewingAngle" on page 231. 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.

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

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, etc.). A GroupNameFieldDefinition Object can be obtained from the Field property of the "FieldObject Object" on page 106 when the specified field is a group name field or from a "GroupNameFieldDefinitions Collection" on page 121 retrieved from the GroupNameFields property of the "Report Object" on page 150.

GroupNameFieldDefinition Object Properties

Property	Description	Read/ Write	Restriction in event handler
GroupNameField Name	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" on page 212. Gets which kind of field (for example, database, summary, formula, etc.).	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" on page 150. 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" on page 212. 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" on page 120 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 (index As Long)	"GroupNameFieldDefinition Object" on page 120. 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" on page 150. 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)).

FieldDefinition Object

The IFieldDefinition Object provides generic properties for the various types of field definition objects.

FieldDefinition Object Properties

Property	Description	Read/ Write	Restriction in event handler
Kind	"CRFieldKind" on page 212. 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" on page 150. 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" on page 212. Gets the field value type.	Read only	None

IReportObject

The IReportObject Object provides generic properties for the various types of report objects.

ReportObject 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	"CRLineStyle" on page 218. 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.
EnableTight Horizontal	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" on page 221. Gets which 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.
LeftLineStyle	"CRLineStyle" on page 218. 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.

Property	Description	Read/ Write	Restriction in event handler
Parent	"Section Object" on page 174. Reference to the parent object.	Read only	Can be written only when formatting idle or active.
RightLineStyle	"CRLineStyle" on page 218. 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	"CRLineStyle" on page 218. 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.

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.
EndSection	"Section Object" on page 174. Gets the end section.	Read only	None
ExtendToBottom OfSection	Boolean. Gets or sets the extend to bottom of section option.	Read / Write	Can be written only when formatting idle.
Kind	"CRObjectKind" on page 221. Gets which 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.
LineColor	OLE_COLOR. Gets or sets the line color.	Read / Write	Can be written only when formatting idle.
LineStyle	"CRLineStyle" on page 218. Gets or sets the line style. See Remarks below.	Read / Write	Can be written only when formatting idle.

Property	Description	Read/Write	Restriction in event handler
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" on page 174. 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

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.
BottomLine Style	"CRLineStyle" on page 218. 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.
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.

Property	Description	Read/ Write	Restriction in event handler
Kind	"CRObjectKind" on page 221. Gets which 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.
LeftLineStyle	"CRLineStyle" on page 218. 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" on page 174. Gets reference to the parent object.	Read only	None
RightLineStyle	"CRLineStyle" on page 218. 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	"CRLineStyle" on page 218. 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" on page 198 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 (index As Long)	"SummaryFieldDefinition Object" on page 198. 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	"IReportObject" on page 122. 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 126

"Delete Method (ObjectSummaryFieldDefinitions Collection)" on page 126

Add Method (ObjectSummaryFieldDefinitions Collection)

Use 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 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

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" on page 218. 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.
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" on page 221. 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	"CRLineStyle" on page 218. 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" on page 174. Gets reference to the parent object.	Read only	None
RightLineStyle	"CRLineStyle" on page 218. 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	"CRLineStyle" on page 218. 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

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" on page 218. 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.
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	"CRObjectKind" on page 221. Gets which 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.

Property	Description	Read/ Write	Restriction in event handler
LeftLineStyle	"CRLineStyle" on page 218. 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" on page 174. 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" on page 218. 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" on page 218. 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 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 methods are discussed in this section:

"SetOleLocation Method (OleObject Object)" on page 129

SetOleLocation Method (OleObject Object)

Use 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 183.

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
IsMissing TotalPageCount	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" on page 135. 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 131

"RenderHTML Method (Page Object)" on page 131

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" on page 225. 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
includeDrillDown Links	Indicates whether or not the HTML page will include hyperlinks for drilling-down on summary data.
pageStyle	"CRHTMLPageStyle" on page 217. Specifies the style of the HTML page to be rendered.

Parameter	Description
toolbarStyle	"CRHTMLToolbarStyle" on page 217. 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" on page 225. 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" on page 217. Gets or sets the image type for EPF format.	Read / Write	Can be written only when formatting idle.
Parent	"Report Object" on page 150. Gets reference to the parent object.	Read only	None
PlaceHolder Options	"CRPlaceHolderType" on page 224. Gets or sets the EPF place holder options.	Read / Write	Can be written only when formatting idle.
ValueFormat Options	"CRValueFormatType" on page 231. 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)

The CreatePageGenerator method returns a "PageGenerator Object" on page 135, 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" on page 135.

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 CRRenderResultType)
```

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	"CRRenderResultType" on page 225. 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 CRRenderResultType)
```

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" on page 225. 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" on page 130, 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
ContainingGroup Name	String. Gets containing group name for out of place subreport view.	Read only	None
ContainingGroup Path	Variant. Gets containing group path for out of place subreport view.	Read only	None
ContainingPage Number	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)" on page 133.	Read only	None
Pages	"Pages Collection" on page 141. Gets the collection of pages for a specified view.	Read only	None

Property	Description	Read/ Write	Restriction in event handler
Parent	“PageEngine Object” on page 132. 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)

Use 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 memberwhere 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" on page 135 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" on page 135.

DrillOnMap Method (PageGenerator Object)

The DrillOnMap method creates a new"PageGenerator Object" on page 135, 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" on page 135.

DrillOnSubreport Method (PageGenerator Object)

The DrillOnSubreport method creates a new "PageGenerator Object" on page 135, 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" on page 135.

Export Method (PageGenerator Object)

Use Export method to obtain an export stream.

Syntax

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

Parameter

Parameter	Description
resultType	"CRRenderResultType" on page 225. Sepcifies the result type.

Returns

Returns the export data stream.

FindText Method (PageGenerator Object)

Use 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" on page 227. 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 memberwhere 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" on page 225. 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 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.
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.

Parameter	Description
baseURL	The URL used to access the report when it is first generated.
resultType	"CRRenderResultType" on page 225. 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" on page 130. 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" on page 130. Gets the collection item based on the 1-based index number.	Read only	None
Parent	"PageGenerator Object" on page 135. 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, etc.). A ParameterFieldDefinition Object is obtain from the Field property of the "FieldObject Object" on page 106, when the specified field is a parameter field or from the "ParameterFieldDefinitions Collection" on page 148 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.
DiscreteOrRange Kind	"CRDiscreteOrRangeKind" on page 210. Gets or sets the parameter value kind (discrete and / or range).	Read / Write	Can be written only when formatting idle.
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.
EnableExclusive Group	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.
EnableMultiple Values	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.
EnableShow DescriptionOnly	Boolean. Gets or sets the show description for pick list option.	Read / Write	Can be written only when formatting idle.
EnableSortBasedOn Desc	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

Property	Description	Read/ Write	Restriction in event handler
IsDefaultValueSet	Boolean. Gets whether the default value has been set.	Read only	None
Kind	"CRFieldKind" on page 212. 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.
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" on page 223. Gets parameter type.	Read only	None
Parent	"Report Object" on page 150. Gets reference to the parent object.	Read only	None
PickListSortMethod	"CRParameterPickListSortMethod" on page 223. Gets or sets the sort method in pick list option.	Read / Write	Can be written only when formatting idle.

Property	Description	Read/ Write	Restriction in event handler
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 formattiong 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” on page 212. 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)

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" on page 225. 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" on page 225. 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” on page 142. Gets the item in the Collection specified by parameter index.	Read only	None
Parent	“Report Object” on page 150. 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 Methods

The following methods are discussed in this section:

"Add Method (ParameterFieldDefinitions Collection)" on page 149

"Delete Method (ParameterFieldDefinitions Collection)" on page 149

Add Method (ParameterFieldDefinitions Collection)

Use Add method to add a "ParameterFieldDefinition Object" on page 142, to the ParameterFieldDefinitions Collection.

Syntax

```
Function Add (parameterName As String, ValueType As CRFieldValueType
    ) As ParameterFieldDefinition
```

Parameters

Parameter	Description
parameterName	Specifies the parameter name.
ValueType	"CRFieldValueType" on page 212. Specifies the value type of the field.

Returns

Returns the ParameterFieldDefinition Object added to the Collection

Delete Method (ParameterFieldDefinitions Collection)

Use Delete method to remove a ParameterFieldDefinition 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, etc.). A PrintingStatus Object is obtained from the PrintingStatus property of the "Report Object" on page 150.

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
NumberOfRecord Printed	Long. Gets the number of records printed.	Read only	None
NumberOfRecordRead	Long. Gets the number of records read.	Read only	None
NumberOfRecord Selected	Long. Gets the number of records selected.	Read only	None
Parent	"Report Object" on page 150. Gets reference to the parent object.	Read only	None
Progress	"CRPrintingProgress" on page 225. 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" on page 68. 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" on page 68. Gets reference to the Application Object that the Report Object is associated with.	Read only	None
Application Name	String. Gets or sets the application name.	Read / Write	None

Areas	"Areas Collection" on page 77. 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.
CanPerform GroupingOn Server	Boolean. Gets whether the report can perform grouping on the server.	Read only	None
CaseInsensitive SQLData	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.
ConvertDate TimeType	"CRConvertDateTimeType" on page 208. 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.
ConvertNull FieldToDefault	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" on page 85. 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" on page 100. Gets reference to ExportOptions Object for the report.	Read only	None
FieldMapping Type	"CRFieldMappingType" on page 212. Gets or sets the field mapping type.	Read / Write	Can be written only when formatting idle.
FormulaFields	"FormulaFieldDefinitions Collection" on page 114. Gets reference to Collection of all the named FormulaFieldDefinitions defined in the Report.	Read only	None
FormulaSyntax	"CRFormulaSyntax" on page 213. Gets or sets report formula syntax.	Read / Write	Can be written only when formatting idle.
GroupName Fields	"GroupNameFieldDefinitions Collection" on page 121. 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" on page 187. 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" on page 225. Gets what kind of report.	Read only	None
LastGetFormula Syntax	"CRFormulaSyntax" on page 213. 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.
MorePrintEngine ErrorMessages	Boolean. Gets or sets the report option that indicates whether to pop up database error dialogs during printing when a Report Engine error occurs.	Read / Write	Can be written only when formatting idle.
NumberOfGroup	Long. Gets the number of groups in the report.	Read only	None
PageEngine	"PageEngine Object" on page 132. Gets reference to the PageEngine object.	Read only	None
PaperOrientation	"CRPaperOrientation" on page 221. 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" on page 221. 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" on page 223. Gets or sets the current printer paper source.	Read / Write	Can be written only when formatting idle.
ParameterFields	"ParameterFieldDefinitions Collection" on page 148. 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" on page 150. 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" on page 224. 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" on page 150. Gets PrintingStatus Object for the report.	Read only	None
RecordSelection Formula	String. Gets or sets record selection formula.	Read / Write	Can be written only when formatting idle.
RecordSortFields	"SortFields Collection" on page 187. Gets a collection of record sort fields.	Read only	None
ReportAlerts	"ReportAlerts Collection" on page 164. 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.
RunningTotal Fields	"RunningTotalFieldDefinitions Collection" on page 172. Gets running total fields collection.	Read only	None
SavePreview Picture	Boolean. Gets or sets save preview picture with report option.	Read/Write	None
Sections	"Sections Collection" on page 185. Gets collection of all the sections in the report.	Read only	None
SQLExpression Fields	"SQLExpressionFieldDefinitions Collection" on page 191. 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" on page 200. 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.
TranslateDos Memos	Boolean. Gets or sets the report option that indicates whether to translate DOS memos.	Read/Write	Can be written only when formatting idle.
TranslateDos Strings	Boolean. Gets or sets the report option that indicates whether to translate DOS strings.	Read/Write	Can be written only when formatting idle.
UseIndexFor Speed	Boolean. Gets or sets the use index for speed during record selection report option.	Read/Write	Can be written only when formatting idle.
VerifyOnEvery Print	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)

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" on page 216. Specifies CRGroupCondition (see table below) indicating the grouping condition (that is, group on any value).
SortDirection	"CRSortDirection" on page 228. Specifies the sort direction for the group.

AddReportVariable Method (Report Object)

Use 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" on page 226. 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 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" on page 208. 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 CancelPrinting method to cancel the printing of a report.

Syntax

```
Sub CancelPrinting ()
```

DeleteGroup Method (Report Object)

Use 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 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 Export method to export reports to a format and destination specified with "ExportOptions Object" on page 100.

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 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 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 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 PrintOut method to print out the specified pages of the report to the printer selected using the “SelectPrinter Method (Report Object)” on page 160. 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.

ReadRecords Method (Report Object)

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

Syntax

```
Sub ReadRecords ()
```

SaveAs Method (Report Object)

Use 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” on page 225. 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 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 me 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.

Report Object Events

The following events are discussed in this section:

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. ReportAlert Object Properties. 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" on page 167. Gets a reference to a collection of all the Report Alert instances created when the Report Alert is run.	Read only	None

Property	Definition	Read/ Write	Restriction in event handler
ConditionFormula	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.
DefaultMessage	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 wether or not the Report Alert is enabled.	Read/ Write	Can be written only when formatting idle
MessageFormula	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" on page 150. 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" on page 163,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" on page 163, in the collection.	Read only	None
Item (index As Long)	"ReportAlert Object" on page 163. 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" on page 150. 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 165

"Delete Method (ReportAlerts Collection)" on page 166

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 wether 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 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

PropertLy	Definition	Read/Write	Restriction in event handler
AlertMessage	String. Gets the message returned at the time the Report Alert was triggered.	Read only	None
Parent	"ReportAlert Object" on page 163. 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" on page 167,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" on page 167, in the collection.	Read only	None
Item (index As Long)	"ReportAlertInstance Object" on page 167. 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" on page 163. 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" on page 77, "FieldObject Object" on page 106, "TextObject Object" on page 204,"Page Object" on page 130, "CrossTabObject Object" on page 82, "SubreportLinks Collection" on page 193, "" on page 77, "GraphObject Object" on page 115, "LineObject Object" on page 123, "MapObject Object" on page 124, "OlapGridObject Object" on page 127. 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" on page 174. 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
Evaluate Condition	"CRRunningTotalCondition" on page 227. Gets evaluate condition.	Read only	None
Evaluate ConditionField	Object. Gets evaluate condition field.	Read only	None
Evaluate ConditionFormula	String. Gets or sets evaluate condition formula.	Read / Write	Can be written only when formatting idle.
Evaluate GroupNumber	Integer. Gets or sets evaluate group number.	Read / Write	Can be written only when formatting idle.
Hierarchical SummaryType	"CRHierarchicalSummaryType" on page 216. Gets or sets wether 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	"CRFieldKind" on page 212. 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	"Report Object" on page 150. 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	"CRRunningTotalCondition" on page 227. Gets the reset condition.	Read only	None
ResetCondition Field	Object. Gets reset condition field.	Read only	None
Reset ConditionFormula	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.

Property	Definition	Read/ Write	Restriction in event handler
RunningTotal FieldName	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.
Summary OperationParameter	Long. Gets or sets summary operation parameter.	Read / Write	Can be written only when formatting idle.
SummaryType	"CRSummaryType" on page 229. 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" on page 212. 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)

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

Syntax

```
Sub SetNoEvaluateCondition ()
```

SetNoResetCondition Method (RunningTotalFieldDefinition Object)

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

Syntax

```
Sub SetNoResetCondition ()
```

SetResetConditionField Method (RunningTotalFieldDefinition Object)

Use 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 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
secondarySummariedField	Specifies the condition field that you want to use.

SetSummarizedField Method (RunningTotalFieldDefinition Object)

Use 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 RunningTotalFieldDefinition**s** 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" on page 168, in the collection using the Item property.

RunningTotalFieldDefinitions Collection Properties

Property	Description	Read/Write	Restriction in event handler
Count	Long. Gets the number of "RunningTotalFieldDefinition Object" on page 168, in the collection.	Read only	None
Item (index As Long)	"RunningTotalFieldDefinition Object" on page 168. 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	"Report Object" on page 150. 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 173

"Delete Method (RunningTotalFieldDefinitions Collection)" on page 173

Add Method (RunningTotalFieldDefinitions Collection)

Use Add method to add a "RunningTotalFieldDefinition Object" on page 168 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" on page 168 member of the Collection.

Delete Method (RunningTotalFieldDefinitions Collection)

Use 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.

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.
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
Parent	“Area Object” on page 74. Gets reference to the parent object.	Read only	None
PrintAtBottomOf Page	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” on page 168. Gets reference to the heterogeneous collection of report objects for the section.	Read only	None
ResetPage NumberAfter	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.

Property	Description	Read/ Write	Restriction in event handler
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)

The AddBlobFieldObject method adds a “BlobFieldObject Object” on page 77 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” on page 77.

AddBoxObject Method (Section Object)

The BoxObject method adds a “” on page 77 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 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 BoxObject will be placed, if not the parent Section.

Returns

Returns a "" on page 77.

AddCrossTabObject Method (Section Object)

The AddCrossTabObject method adds a "CrossTabObject Object" on page 82 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" on page 82.

AddFieldObject Method (Section Object)

The AddFieldObject method adds a "FieldObject Object" on page 106 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 “FieldObject Object” on page 106.

AddGraphObject Method (Section Object)

The AddGraphObject method adds a “GraphObject Object” on page 115 (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” on page 213. 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” on page 115.

AddLineObject Method (Section Object)

The AddLineObject method adds a “LineObject Object” on page 123 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.
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” on page 123.

AddPictureObject Method (Section Object)

The AddLineObject method adds a picture object from an image in the form of an “OleObject Object” on page 128 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 a “OleObject Object” on page 128.

AddSpecialVarFieldObject Method (Section Object)

The SpecialVarFieldObject Method adds a SpecialVar "FieldObject Object" on page 106 to the Section Object.

Syntax

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

Parameters

Parameter	Description
specialVarType	"CRSpecialVarType" on page 228. 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 "FieldObject Object" on page 106.

AddSubreportObject Method (Section Object)

The SubReportObject method adds a "SubreportObject Object" on page 195 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" on page 195.

AddSummaryFieldObject Method (Section Object)

The AddSummaryField method adds a summary "FieldObject Object" on page 106 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" on page 229. 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.
[secondSummary FieldOrFactor]	Specifies the second summary field or factor, if required by the summary type.

Returns

Returns a "FieldObject Object" on page 106.

AddTextObject Method (Section Object)

The AddTextObject method adds a "TextObject Object" on page 204 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, etc.). If FALSE, formatting characters in the text string will be ignored.

Returns

Returns a “FieldObject Object” on page 106.

AddUnboundFieldObject Method (Section Object)

The AddUnboundFieldObject method adds an unbound “FieldObject Object” on page 106 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” on page 212. 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 “FieldObject Object” on page 106.

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)

The ImportSubreport method imports a "SubreportObject Object" on page 195 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" on page 195.

Section Object Events

The following events are discussed in this section:

"Format Event (Section Object)" on page 183

Format Event (Section Object)

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" on page 150 or "Area Object" on page 74. 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" on page 174. 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", etc.).	Read only	None
Parent	"Report Object" on page 150 or "Area Object" on page 74. 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 186

"Delete Method (Sections Collection)" on page 186

Add Method (Sections Collection)

Use Add method to add a "Section Object" on page 174 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" on page 174 member of the Sections Collection.

Delete Method (Sections Collection)

Use Delete method to remove a "Section Object" on page 174 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" on page 150. Gets reference to the parent object.	Read only	None
SortDirection	"CRSortDirection" on page 228. 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" on page 187. 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" on page 150. 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 188

"Delete Method (SortFields Collection)" on page 188

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	"FieldDefinition Object" on page 121. Specifies the field definition or field name.
SortDirection	"CRSortDirection" on page 228. 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 (i.e., last modification date, print date, etc.). A SpecialVarFieldDefinition Object is obtained from the Field property of the "FieldObject Object" on page 106.

SpecialVarFieldDefinition Object Properties

Property	Description	Read/Write	Restriction in event handler
Kind	"CRFieldKind" on page 212. 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	"Report Object" on page 150. 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	"CRSpecialVarType" on page 228. Gets what the type of special field (for example, ReportTitle, PageNumber, etc.).	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" on page 212. 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" on page 212. 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" on page 150. 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.
SQLExpression FieldName	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" on page 212. 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 191

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" on page 190 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" on page 190. 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" on page 150. 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 192

"Delete Method (SQLExpressionFieldDefinitions Collection)" on page 192

Add Method (SQLExpressionFieldDefinitions Collection)

Use Add method to add a "SQLExpressionFieldDefinition Object" on page 190 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" on page 190 member of the Collection.

Delete Method (SQLExpressionFieldDefinitions Collection)

Use Delete method to remove a "SQLExpressionFieldDefinition Object" on page 190 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.

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	"FieldDefinition Object" on page 121. Gets the main report field to which the subreport field is linked.	Read only	None
Parent	"SubreportObject Object" on page 195. Gets the subreport link's parent object.	Read only	None
SubReportField	"FieldDefinition Object" on page 121. Gets the subreport field.	Read only	None

SubreportLinks Collection

The SubreportLinks Collection contains the "SubreportLink Object" on page 193 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" on page 193. Gets an item from the Collection.	Read only	None
Parent	"SubreportObject Object" on page 195. 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)

Use Add method to add a "SubreportLink Object" on page 193 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" on page 193 member of the Collection.

Delete Method (SubreportLinks Collection)

Use Delete method to remove a "SubreportLink Object" on page 193 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, etc.).

SubreportObject 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	"CRLineStyle" on page 218. 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
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.
KeepTogether	Boolean. Gets or sets the keep object together option.	Read / Write	Can be written only when formatting idle or active.
Kind	"CRObjectKind" on page 221. 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" on page 218. 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
Links	"SubreportLinks Collection" on page 193. 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" on page 174. Gets reference to the parent object.	Read only	None
RightLineStyle	"CRLineStyle" on page 218. 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" on page 218. 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 196

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" on page 195.

ReImportSubreport Method (Subreport Object)

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.

Sample

```
'Boolean value to hold the result of the ReimportSubreport method
Dim bReimported As Boolean

'Call ReImportSubReport
Report.Subreport1.ReimportSubreport bReimported

'If bReimported is True the subreport re-imported successfully
'If bReimported is False the subreport failed to re-import
If bReimported Then
    MsgBox "Subreport re-imported successfully"
Else
    MsgBox "Subreport failed to re-import"
    Exit Sub
End If

'Refresh the viewer to display the updated subreport.
'Set the parameter to True if you want to update the data
'or False if you only want to refresh the report
CRViewer1.RefreshEx False
```

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" on page 74. Gets the area pair that the summary is in.	Read only	None
ForCrossTab	Boolean. Gets for-cross-tab.	Read only	None
HeaderArea	"Area Object" on page 74. Gets the area pair that the summary is in.	Read only	None
Hierarchical SummaryType	"CRHierarchicalSummaryType" on page 216. Gets or sets wether 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" on page 212. 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" on page 150. 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.
Secondary SummarizedField	Object. Gets the secondary summarized field.	Read only	None
SummarizedField	Object. Gets the summarized field.	Read only	None
SummaryOperationParameter	Long. Gets or sets the summary operation parameter.	Read / Write	Can be written only when formatting idle.

Property	Description	Read/ Write	Restriction in event handler
SummaryType	"CRSummaryType" on page 229. Gets or sets the type of summary (for example, sum, average, etc.).	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" on page 212. 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 199

"SetSummarizedField Method (SummaryFieldDefinition Object)" on page 199

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" on page 198. 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" on page 150. 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 200

"Delete Method (SummaryFieldDefinitions Collection)" on page 201

Add Method (SummaryFieldDefinitions Collection)

Use Add method to add a "SummaryFieldDefinition Object" on page 198 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.
SummaryType	"CRSummaryType" on page 229. Specifies the summary type.
[secondSummaryFieldOr Factor]	Specifies the second summary field or factor.

Returns

Returns a "SummaryFieldDefinition Object" on page 198 member of the Collection.

Delete Method (SummaryFieldDefinitions Collection)

Use delete method to delete a "SummaryFieldDefinition Object" on page 198 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

Property	Description	Read/Write	Restriction in event handler
DestinationFields	"DatabaseFieldDefinitions Collection" on page 93. Gets reference to table link destination field definitions Collection.	Read only	None
DestinationTable	"DatabaseTable Object" on page 93. Gets reference to the table link destination table.	Read only	None
IndexUsed	Integer. Gets table link IndexUsed	Read only	None
JoinType	"CRLinkJoinType" on page 218. Gets the table link join type.	Read only	None
LookupType	"CRLinkLookUpType" on page 219. Gets table link lookup type.	Read only	None
Parent	"Database Object" on page 85. Gets reference to the parent object.	Read only	None
PartialMatch Enabled	Boolean. Gets the table link partial-match enabled option.	Read only	None
SourceFields	"DatabaseFieldDefinition Object" on page 92. Gets the table link source field definitions.	Read only	None
SourceTable	"DatabaseTable Object" on page 93. Gets table link source table.	Read only	None

TableLinks Collection

The TableLinks Collection contains the "TableLink Object" on page 202 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" on page 202. Gets an item from the Collection.	Read only	None
Parent	"Database Object" on page 85. 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 203

"Delete Method (TableLinks Collection)" on page 203

Add Method (TableLinks Collection)

Use Add method to add a "TableLink Object" on page 202 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" on page 93. Specifies the source database table.
pDestTable	"DatabaseTable Object" on page 93. Specifies the destination database table.
srcFields	Specifies the source fields.
destFields	Specifies th destination fields.
JoinType	"CRLinkJoinType" on page 218. Specifies the link join type.
LookupType	"CRLinkLookUpType" on page 219. Specifies the link lookup type.
PartialMatchEnabled	Specifies whether the PartialMatchEnabled option has been set.
indexInUse	Specifies index in use.

Returns

Returns a "TableLink Object" on page 202 member of the Collection.

Delete Method (TableLinks Collection)

Use Delete method to remove a "TableLink Object" on page 202 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

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" on page 218. 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.
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	"CRAlignment" on page 207. Gets or sets the horizontal alignment.	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 or active.
Kind	"CRObjectKind" on page 221. 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.	Read / Write	Can be written only when formatting idle.

Property	Description	Read/Write	Restriction in event handler
LeftLineStyle	"CRLineStyle" on page 218. 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 formattin idle.
LineSpacingType	"CRLineSpacingType" on page 218. Gets the line spacing type.	Read only	
MaxNumberOf Lines	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" on page 174. 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" on page 218. 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.
SuppressIf Duplicated	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.
TextRotation Angle	"CRRotationAngle" on page 226. Gets or sets text rotation angle.	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	"CRLineStyle" on page 218. 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 206

"SetLineSpacing Method (TextObject Object)" on page 206

SetLineSpacing Method (TextObject Object)

Use the SetLineSpacing method to set the line spacing for a "TextObject Object" on page 204. The information passed will be used during formatting.

Syntax

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

Parameters

Parameter	Description
LineSpacing	Specifies the line spacing.
LineSpacingType	"CRLineSpacingType" on page 218. 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.

Enumerated Types

CRAlignment

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

CRBooleanOutputType

Constant	Value
crOneOrZero	4
crTOrF	1
crTrueOrFalse	0
crYesOrNo	2
crYOrN	3

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
crHijriCalendar	6
crJapaneseCalendar	3
crKoreanCalendar	5
crTaiwaneseCalendar	4
crThaiCalendar	7

CRDateEraType

Constant	Value
crLongEra	1
crNoEra	2
crShortEra	0

CRDateOrder

Constant	Value
crDayMonthYear	1
crMonthDayYear	2
crYearMonthDay	0

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

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
crEFTExactRichText	35
crEFTExcel50	21
crEFTExcel50Tabular	22
crEFTExcel70	27
crEFTExcel70Tabular	28
crEFTExcel80	29
crEFTExcel80Tabular	30
crEFTExplorer32Extend	25
crEFTHTML32Standard	24
crEFTHTML40	32
crEFTLotus123WK1	12
crEFTLotus123WK3	13
crEFTLotus123WKS	11
crEFTNoFormat	0
crEFTODBC	23
crEFTPaginatedText	10
crEFTPortableDocFormat	31
crEFTRecordStyle	3
crEFTReportDefinition	34
crEFTTabSeparatedText	9
crEFTTabSeparatedValues	6
crEFTText	8
crEFTWordForWindows	14
crEFTXML	36

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
crBlobField	15
crBooleanField	9
crChartField	21
crCurrencyField	8
crDateField	10
crDateTimeField	16
crIconField	18
crInt16sField	3
crInt16uField	4
crInt32sField	5
crInt32uField	6
crInt8sField	1
crInt8uField	2

Constant	Value
crNumberField	7
crOleField	20
crPersistentMemoField	14
crPictureField	19
crStringField	12
crTimeField	11
crTransientMemoField	13
crUnknownField	22

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
crDualAxisBubbleGraph	91
crFaked3DAbsoluteAreaGraph	23
crFaked3DPercentAreaGraph	25
crFaked3DPercentBarGraph	5
crFaked3DRegularPieGraph	31
crFaked3DSideBySideBarGraph	3
crFaked3DStackedAreaGraph	24
crFaked3DStackedBarGraph	4
crHighLowDualAxisGraph	101 Obsolete.
crHighLowGraph	100
crHighLowOpenCloseDualAxisGraph	105 Obsolete.
crHighLowOpenCloseGraph	104
crHighLowOpenDualAxisGraph	103 Obsolete.
crHighLowOpenGraph	102 Obsolete.
crLineGraphWithMarkers	13
crMultipleDoughnutGraph	41
crMultiplePieGraph	32
crMultipleProportionalDoughnut Graph	42
crMultipleProportionalPieGraph	33
crPercentageLineGraph	12
crPercentageLineGraphWithMarkers	15
crPercentAreaGraph	22
crPercentBarGraph	2
crRadarDualAxisGraph	82
crRegularBubbleGraph	90

Constant	Value
crRegularDoughnutGraph	40
crRegularLineGraph	10
crRegularPieGraph	30
crRegularRadarGraph	80
crSideBySideBarGraph	0
crStackedAreaGraph	21
crStackedBarGraph	1
crStackedLineGraph	11
crStackedLineGraphWithMarkers	14
crStackedRadarGraph	81
crThreeDCutCornersGraph	53
crThreeDOctagonGraph	52
crThreeDPyramidGraph	51
crThreeDRegularGraph	50
crThreeDSurfaceHoneycombGraph	62
crThreeDSurfaceRegularGraph	60
crThreeDSurfaceWithSidesGraph	61
crUnknownGraph	1000
crXyScatterDualAxisGraph	71
crXyScatterDualAxisWithLabelsGraph	73
crXyScatterGraph	70
crXyScatterWithLabelsGraph	72

CRGridlineType

Constant	Value
crMajorAndMinorGridlines	3
crMajorGridlines	2
crMinorGridlines	1
crNoGridlines	0

CRGroupCondition

Constant	Value
crGCAnnually	7
crGCAnyValue	14
crGCBiweekly	2
crGCByAMPM	18
crGCByHour	17
crGCByMinute	16
crGCBySecond	15
crGCDaily	0
crGCEveryNo	11
crGCEveryYes	10
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
Const crNoHour	2
Const crNumericHour	0
Const 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

CRLeadingDayPosition

Constant	Value
crLeadingDayOfWeek	0
crTrailingDayOfWeek	1

CRLeadingDayType

Constant	Value
crLongLeadingDay	1
crNoLeadingDay	2
crShortLeadingDay	0

CRLegendPosition

Constant	Value
crPlaceBottomCenter	1
crPlaceLeft	4
crPlaceRight	3
crPlaceTopCenter	2
crPlaceUpperRight	0

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
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

CRObjectKind

Constant	Value
crBlobFieldObject	9
crBoxObject	4
crCrossTabObject	8
crFieldObject	1
crGraphObject	7
crLineObject	3
crMapObject	10
crOlapGridObject	11
crOleObject	6
crSubreportObject	5
crTextObject	2

CRPaperOrientation

Constant	Value
crDefaultPaperOrientation	0
crLandscape	2
crPortrait	1

CRPaperSize

Constant	Value
crDefaultPaperSize	0
crPaper10x14	16
crPaper11x17	17
crPaperA3	8
crPaperA4	9
crPaperA4Small	10
crPaperA5	11
crPaperB4	12

crPaperB5	13
crPaperCsheet	24
crPaperDsheet	25
crPaperEnvelope10	20
crPaperEnvelope11	21
crPaperEnvelope12	22
crPaperEnvelope14	23
crPaperEnvelope9	19
crPaperEnvelopeB4	33
crPaperEnvelopeB5	34
crPaperEnvelopeB6	35
crPaperEnvelopeC3	29
crPaperEnvelopeC4	30
crPaperEnvelopeC5	28
crPaperEnvelopeC6	31
crPaperEnvelopeC65	32
crPaperEnvelopeDL	27
crPaperEnvelopeItaly	36
crPaperEnvelopeMonarch	37
crPaperEnvelopePersonal	38
crPaperEsheet	26
crPaperExecutive	7
crPaperFanfoldLegalGerman	41
crPaperFanfoldStdGerman	40
crPaperFanfoldUS	39
crPaperFolio	14
crPaperLedger	4
crPaperLegal	5
crPaperLetter	1
crPaperLetterSmall	2
crPaperNote	18
crPaperQuarto	15
crPaperStatement	6
crPaperTabloid	3

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
crCustomLayout	2
crPercentLayout	0

CRPieSize

Constant (numeric order)	Value
crMaximumPieSize	0
crLargePieSize	16
crAveragePieSize	32
crSmallPieSize	48
crMinimumPieSize	64

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
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

Constant (numeric order)	Value
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

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

CRSummaryType

Constant	Value
crSTAverage	1
crSTCount	6
crSTDCorrelation	10
crSTDCovariance	11
crSTDistinctCount	9
crSTDMedian	13
crSTDMode	17
crSTDNthLargest	15
crSTDNthMostFrequent	18
crSTDNthSmallest	16
crSTDPercentage	19
crSTDPercentile	14
crSTDWeightedAvg	12
crSTMaximum	4
crSTMinimum	5
crSTPopStandardDeviation	8
crSTPopVariance	7
crSTSampleStandardDeviation	3
crSTSampleVariance	2
crSTSum	0

CRTableDifferences

Constant	Value
crTDOK	0x00000000
crTDDatabaseNotFound	0x00000001
crTDServerNotFound	0x00000002
crTDServerNotOpened	0x00000004
crTDAliasChanged	0x00000008
crTDIndexesChanged	0x00000010
crTDDriverChanged	0x00000020
crTDDictionaryChanged	0x00000040
crTDFileTypeChanged	0x00000080

Constant	Value
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

CRTopOrBottomNGroupSortOrder

Constant	Value
crAllGroupsSorted	1
crAllGroupsUnsorted	0
crBottomNGroups	3
crTopNGroups	2

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
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 in Version 8:

- 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 Seagate Software Developer Zone web site at `http://www.seagatesoftware.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 236.

“Using the CRViewer object” on page 236.

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.

1. First, you must verify that a reference to the Report Viewer control exists in your project. 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 Report Viewer*.

   Note: If you do not see the Crystal Report Report Viewer control in the list, use the Browse button to locate the CRVIEWER.DLL component in the `C:\Program Files\Seagate Software\Viewers\ActiveXViewer` directory.
3. If the check box next to the Report Viewer control is not toggled on, toggle it on now.
4. Click OK, and the CRViewer control will appear 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

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 fort 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(GroupNameList As Variant, _
      ByVal DrillType As CRVIEWERLibCtl.CRDrillType, UseDefault As
Boolean)
      MsgBox "You're drilling down on the " & GroupNameList(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 on a chart, on a map, or on 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 239.) 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
- EnableGroupTree
- EnableNavigationControls
- EnablePrintButton

- EnableProgressControl
- EnableRefreshButton
- EnableSearchControl
- 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. All controls and features are, by default, enabled.

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” on page 241, and “Connecting from VBScript” on page 242, 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 (\SeagateSoftware\Viewers\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.
3. 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 on the Report Viewer Beans name (Crystal Report Viewer) in the ToolBox palette.
2. Click on 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.
4. 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.
5. 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 on it. Your default browser should display the Report Viewer and the report.

The minimum required to actually run the application using the bean is:

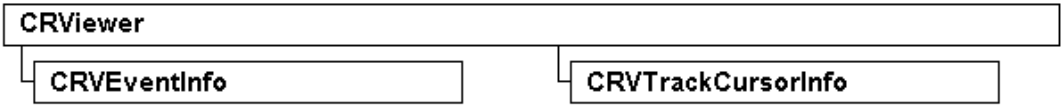
- the html file which references the applet class file
- the extracted ReportViewerBean.jar file and any supporting jar files
- the applet class file.

Report Viewer Object Model 5

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)" on page 268. 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 or sets 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	"CRObjectType (CRViewerLib)" on page 269. Gets the object type.	Read-only

CRVEventInfo Object Methods

The following methods are discussed in this section:

"GetFields Method (CRVEventInfo Object)" on page 247

GetFields Method (CRVEventInfo Object)

Use 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 ICrystalReportViewer3.

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
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

Property	Description	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)” on page 266. 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)

Use 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 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 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 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 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 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 PrintReport method to initiate printing of the report in the current view.

Syntax

```
Sub PrintReport ()
```

Refresh Method (CRViewer Object)

Use 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 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 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 ShowFirstPage method to display the first page of the report.

Syntax

```
Sub ShowFirstPage ()
```

ShowGroup Method (CRViewer Object)

Use 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 ShowLastPage method to display the last page of the report.

Syntax

```
Sub ShowLastPage ()
```

ShowNextPage Method (CRViewer Object)

Use ShowNextPage method to display the next page of the report.

Syntax

```
Sub ShowNextPage ()
```

ShowNthPage Method (CRViewer Object)

Use 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 ShowPreviousPage method to display the previous page of the report.

Syntax

```
Sub ShowPreviousPage ()
```

ViewReport Method (CRViewer Object)

Use ViewReport method to display the report.

Syntax

```
Sub ViewReport ()
```

Zoom Method (CRViewer Object)

Use 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)

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)" on page 247, 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.

DblClicked Event (CRViewer Object)

The DblClicked event occurs when an object is double clicked.

Syntax

```
Event DblClicked ( 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)" on page 247, 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)" on page 269. 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)" on page 269. 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	"CRLoadingType (CRViewerLib)" on page 269. Specifies what type of object the drill event occurred on (for example, graph, group tree, map, etc.).
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 (
    selctionFormula 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 (
    selctionFormula 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)" on page 269. 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)” on page 247.

CRVTrackCursorInfo Object Properties

Property	Description	Read/Write
DetailAreaCursor	“CRTrackCursor (CRViewerLib)” on page 270. Gets or sets the DetailAreaCursor type.	Read / Write
DetailAreaField Cursor	“CRTrackCursor (CRViewerLib)” on page 270. Gets or sets the DetailAreaFieldCursor type.	Read / Write
GraphCursor	“CRTrackCursor (CRViewerLib)” on page 270. Gets or sets the GraphCursor type.	Read / Write
GroupAreaCursor	“CRTrackCursor (CRViewerLib)” on page 270. Gets or sets the GroupAreaCursor type.	Read / Write
GroupAreaField Cursor	“CRTrackCursor (CRViewerLib)” on page 270. 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 267

"AddParameterEx Method (WebReportSource Object)" on page 268

AddParameter Method (WebReportSource Object)

Use 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:

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
LoadingQueryInfo	3
LoadingTotaller	2

CRObjectType (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)

Constant	Value
crPageFooterSection	206 (&HCE)
crPageHeaderSection	205 (&HCD)
crPromptingVarFieldType	9
crReportFooterSection	204 (&HCC)
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

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

Property	Description	Read(r), Write(w), Bound(b), Constrained(c)
currentMessage	string. The message currently displayed in the status area of the toolbar.	r,b
currentPageNumber	int. 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 *exportingPossible*), 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 *showGroupTree*).	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 *printingPossible*) 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
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	int. Indicates to the highest numbered page currently available in the report. This may or may not be the final page. (refer to property *lastPageNumberKnown*).	r,b

Property	Description	Read(r), Write(w), Bound(b), Constrained(c)
lastPageNumber Known	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 int to 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 *hasGroupTree* 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)

Crystal Report Engine 6

The Crystal Report Engine API is a powerful development tool for your reporting needs. In this chapter you will find detailed information on the functions, structures and constants of the Crystal Report Engine API. Syntax for the functions and structures is provided for C, Microsoft Visual Basic, and Delphi. In addition you will find a section on obsolete functions, structures, and constants, including a list of obsolete calls with the applicable replacement calls.

Print Engine Functions

The Print Engine functions are listed alphabetically in this section.

PEAddParameterCurrentRange

Use PEAddParameterCurrentRange to add a parameter range to the specified parameter field of a report. Note that these parameter field capabilities are not currently supported in Web Viewers. See "Working with Parameter Values and Ranges" on page 45.

C Syntax

```
BOOL CRPE_API PEAddParameterCurrentRange (
        short printJob,
        const char FAR *parameterFieldName,
        const char FAR *reportName,
        PEValueInfo FAR *rangeStart,
        PEValueInfo FAR *rangeEnd,
        short rangeInfo );
```

Parameters

printJob	Specifies the print job to which you want to add a parameter current range.
parameterField Name	Specifies a pointer to the string containing the parameter field name.
reportName	Specifies a pointer to the string containing the report name. See Remarks below.
rangeStart	Specifies a pointer to "PEValueInfo" on page 516, which contains the lower bound of the value range.
rangeEnd	Specifies a pointer to "PEValueInfo" on page 516, which contains the upper bound of the value range.
rangeInfo	Use this bitwise value to indicate whether the upper and/or lower bound(s) in the range should be added. Use one or more of the "Range Info Constants" on page 559.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

Regarding parameter reportName:

- For the main report, pass an empty string ("").
- For a subreport, pass the file path and name of the subreport as a NULL-terminated string.

VB Syntax

```
Declare Function PEAddParameterCurrentRange Lib "crpe32.dll" ( _
    ByVal printJob As Integer, ByVal parameterFieldName As String, _
    ByVal reportName As String, rangeStart As PEValueInfo, _
    rangeEnd As PEValueInfo, ByVal rangeInfo As Integer ) As Integer
```

Delphi Syntax

```
procedure PEAddParameterCurrentRange (
    printJob: smallint;
    const parameterFieldName: PChar;
    const reportName: PChar;
    var rangeStart: PEValueInfo;
    var rangeEnd: PEValueInfo;
    rangeInfo: smallint;
    ): BOOL stdcall;
```

PEAddParameterCurrentValue

Use PEAddParameterCurrentValue to add a value to the specified parameter field of a report. Note that these parameter field capabilities are not currently supported in Web Viewers. See “Working with Parameter Values and Ranges” on page 45.

C Syntax

```
BOOL CRPE_API PEAddParameterCurrentValue (
        short printJob,
        const char FAR *parameterFieldName,
        const char FAR *reportName,
        PEValueInfo FAR *currentValue );
```

Parameters

printJob	Specifies the print job to which you want to add a parameter current value.
parameterFieldName	Specifies a pointer to the string containing the parameter field name.
reportName	Specifies a pointer to the string containing the report name. See Remarks below.
currentValue	Specifies a pointer to “PEValueInfo” on page 516, which will contain the value to be added to the parameter field. Use PE_VI_NOVALUE to indicate no value when not NULL (for example, PEValueInfo.currentValue = PE_VI_NOVALUE).

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

Regarding parameter reportName:

- For the main report, pass an empty string ("").
- For a subreport, pass the file path and name of the subreport as a NULL-terminated string.

This call will succeed when called with a ranged parameter. It will add a range with the start and end points equal to the given value to the Current Range list (if there are multiple ranges, otherwise it will replace the current range).

VB Syntax

```
Declare Function PEAddParameterCurrentValue Lib "crpe32.dll" ( _
    ByVal printJob As Integer, ByVal parameterFieldName As String, _
    ByVal reportName As String, currentValue As PEValueInfo ) As Integer
```

Delphi Syntax

```
procedure PEAddParameterCurrentValue (
    printJob: smallint;
    const parameterFieldName: PChar;
    const reportName: PChar;
    var currentValue: PEValueInfo
    ): BOOL stdcall;
```

PEAddParameterDefaultValue

Use PEAddParameterDefaultValue to add a value to the group of default values for the specified parameter in a report.

C Syntax

```
BOOL CRPE_API PEAddParameterDefaultValue (
        short printJob,
        const char FAR *parameterFieldName,
        const char FAR *reportName,
        PEValueInfo FAR *valueInfo );
```

Parameters

printJob	Specifies the print job to which you want to add a parameter default value.
parameterField Name	Specifies a pointer to the name of the parameter field to which the default value will be added.
reportName	Specifies a pointer to the report name. See Remarks below.
valueInfo	Specifies a pointer to “PEValueInfo” on page 516 which will contain the added default value.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

Regarding parameter reportName:

- For the main report, pass an empty string ("").
- For a subreport, pass the file path and name of the subreport as a NULL-terminated string.

VB Syntax

```
Declare Function PEAddParameterDefaultValue Lib "crpe32.dll" ( _
    ByVal printJob As Integer, ByVal parameterFieldName As String, _
    ByVal reportName As String, valueInfo As PEValueInfo ) As Integer
```

Delphi Syntax

```
procedure PEAddParameterDefaultValue (
    printJob: smallint;
    const parameterFieldName: PChar;
    const reportName: PChar;
    var valueInfo: PEValueInfo
    ): BOOL stdcall;
```

PECancelPrintJob

Use PECancelPrintJob to cancel the printing of a report. This function can be tied to a control that allows the user to cancel a print job in progress. You can use this command as a replacement for the Cancel button when “PEShowPrintControls” on page 447, disables the Print Control buttons.

C Syntax

```
void CRPE_API PECancelPrintJob (
        short printJob );
```

Parameter

printJob	Specifies the print job that you want to cancel.

Returns

Void.

VB Syntax

```
Declare Sub PECancelPrintJob Lib “crpe32.dll” (ByVal printJob As
Integer)
```

Delphi Syntax

```
procedure PECancelPrintJob (
    printJob: Word
    )stdcall;
```

dBASE for Windows Syntax

```
EXTERN CVOID PECancelPrintJob (CWORD) CRPE.DLL
```

PECanCloseEngine

Use PECanCloseEngine to determine whether or not the Crystal Report Engine can be closed. Use this function before calling “PECloseEngine” on page 287, to verify that the engine is no longer processing print jobs. If the Crystal Report Engine closes while a print job is still running, an error can occur in your application or on the user’s system.

C Syntax

```
BOOL CRPE_API PECanCloseEngine (void);
```

Returns

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

VB Syntax

```
Declare Function PECanCloseEngine Lib “crpe32.dll” () As Integer
```

Delphi Syntax

```
function PECanCloseEngine:
    Bool stdcall;
```

dBASE for Windows Syntax

```
EXTERN CLOGICAL PECanCloseEngine () CRPE.DLL
```

PECheckFormula

Use to check the text of a named formula for validity. Use this function to check a named formula for errors.

C Syntax

```
BOOL CRPE_API PECheckFormula (
        short printJob,
        const char FAR *formulaName );
```

Parameters

printJob	Specifies the print job containing the named formula that you want to check.
formulaName	Specifies a pointer to the name of the formula that you want to check for errors.

Returns

- TRUE if the formula is correct.
- FALSE if the formula has an error.

Remarks

- When specifying the name of the formula, do not use the @ symbol before the name. The @ symbol is only used by Crystal Reports to separate a formula field from other types of fields.
- PECheckFormula works like the Check button that appears in the Crystal Reports Formula Editor.

VB Syntax

```
Declare Function PECheckFormula Lib "crpe32.dll" (ByVal printJob As
Integer, ByVal FormulaName As String) As Integer
```

Delphi Syntax

```
function PECheckFormula (
    printJob: Word;
    formulaName: PChar
    ): Bool stdcall;
```

dBASE for Windows Syntax

```
EXTERN CLOGICAL PECheckFormula (CWORD, CSTRING) CRPE.DLL
```

PECheckGroupSelectionFormula

Use PECheckGroupSelectionFormula to check the text of the report's group selection formula for errors. Use this function when the group selection formula in a report has changed and you need to check the new group selection formula.

C Syntax

```
BOOL CRPE_API PECheckGroupSelectionFormula (
        short printJob );
```

Parameter

printJob	Specifies the print job for the report containing the group selection formula that you want to check.

Returns

- TRUE if the selection formula does not have an error.
- FALSE if the selection formula contains an error. See Remarks below.

Remarks

- You can use PECheckGroupSelectionFormula to provide your users with the functionality of the Check button that appears in the Crystal Reports Formula Editor.
- If PECheckGroupSelectionFormula returns FALSE, you can provide the associated error text with calls to “PEGetErrorCode” on page 304; “PEGetErrorText” on page 305; and “PEGetHandleString” on page 318.

VB Syntax

```
Declare Function PECheckGroupSelectionFormula Lib “crpe32.dll” (ByVal
printJob As Integer) As Integer
```

Delphi Syntax

```
function PECheckGroupSelectionFormula (
    printJob: Word
    ): Bool stdcall;
```

dBASE for Windows Syntax

```
EXTERN CLOGICAL PECheckGroupSelectionFormula (CWORD) CRPE.DLL
```

PECheckNthTableDifferences

Use PECheckNthTableDifferences to retrieve database table differences. This function is not implemented for reports based on a dictionary.

C Syntax

```
BOOL CRPE_API PECheckNthTableDifferences (
        short printJob,
        short tableN,
        PETableDifferenceInfo FAR *tabledifferenceinfo );
```

Parameters

printJob	Specifies the print job for the report containing the database table that you want to check.
tableN	Specifies the 0-based number of the table for which you want to retrieve table location information. The first table is table 0. The last table is N-1.
tabledifferenceinfo	Specifies a pointer to “PETableDifferenceInfo” on page 508, which will contain the information retrieved.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.
- Returns “Error Codes” on page 545, PE_ERR_NOTIMPLEMENTED if the specified report was based on a dictionary.

VB Syntax

```
Declare Function PECheckNthTableDifferences Lib "crpe32.dll" (
    ByVal printJob As Integer, ByVal tableN As Integer,
    tabledifferenceinfo As PETableDifferenceInfo ) As Integer
```

Delphi Syntax

```
function PECheckNthTableDifferences (
  printJob                : Smallint;
  tableN                  : Smallint;
  var tabledifferenceinfo : PETableDifferenceInfo): Bool; {$ifdef WIN32}
        stdcall; {$endif}
```

PECheckSelectionFormula

Use PECheckSelectionFormula to check the text of the report's record selection formula for errors. Use this function whenever the record selection formula has been changed and you want to check the formula for syntax errors.

C Syntax

```
BOOL CRPE_API PECheckSelectionFormula (
        short printJob );
```

Parameter

printJob	Specifies the print job containing the record selection formula that you want to check.

Returns

- TRUE if the selection formula does not have an error.
- FALSE if the selection formula contains an error. See Remarks below.

Remarks

- You can use PECheckSelectionFormula to provide your users with the functionality of the Check button that appears in the Crystal Reports Formula Editor.
- If PECheckSelectionFormula returns FALSE, you can provide the associated error text with calls to “PEGetErrorCode” on page 304; “PEGetErrorText” on page 305; and “PEGetHandleString” on page 318.

VB Syntax

```
Declare Function PECheckSelectionFormula Lib “crpe32.dll” (ByVal
printJob As Integer) As Integer
```

Delphi Syntax

```
function PECheckSelectionFormula (
    printJob: Word
     ): Bool stdcall;
```

dBASE for Windows Syntax

```
EXTERN CLOGICAL PECheckSelectionFormula (CWORD) CRPE.DLL
```

PECheckSQLExpression

Use to check the text of the specified SQL expression for errors. Use this function whenever the SQL expression has been changed and you want to check the formula for syntax errors.

C Syntax

```
BOOL CRPE_API PECheckSQLExpression (
        short printJob,
        const char FAR *expressionName );
```

Parameters

printJob	Specifies the print job containing the SQL expression that you want to check.
expressionName	Specifies a pointer to the name of the SQL expression that you want to check.

Returns

- TRUE if the SQL expression does not have an error.
- FALSE if the SQL expression contains an error.

Remarks

You must be connected to the database to use this function.

VB Syntax

```
Declare Function PECheckSQLExpression Lib "crpe32.dll" (ByVal printJob
As Integer, ByVal expressionName As String) As Integer
```

Delphi Syntax

```
function PECheckSQLExpression (
    printJob: smallint;
    const expressionName: PChar
    ): Bool stdcall;
```

PEClearParameterCurrentValuesAndRanges

Use PEClearParameterCurrentValuesAndRanges to clear the specified parameter field of all current values and ranges.

C Syntax

```
BOOL CRPE_API PEClearParameterCurrentValuesAndRanges (
        short printJob,
        const char FAR *parameterFieldName,
        const char FAR *reportName );
```

Parameters

printJob	Specifies the print job for which you want to clear specified parameter fields.
parameterField Name	Specifies a pointer to the string containing the parameter field name.
reportName	Specifies a pointer to the string containing the report name. See Remarks below

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEClearParameterCurrentValuesAndRanges Lib "crpe32.dll"
_
    ( ByVal printJob As Integer, ByVal parameterFieldName As String, _
    ByVal reportName As String ) As Integer
```

Delphi Syntax

```
function PEClearParameterCurrentValuesAndRanges (
    printJob: smallint;
    const parameterFieldName: PChar;
    const reportName: PChar
    ): BOOL stdcall;
```

PECloseEngine

Use PECloseEngine to terminate the Crystal Report Engine. All printing is stopped and all windows are closed. This function stops the Crystal Report Engine from sending output, but the report may continue to print from data remaining in the spooler.

C Syntax

```
void CRPE_API PECloseEngine (void);
```

Returns

Void.

Remarks

- Once this function has been called, no other Crystal Report Engine functions can be called except “PEOpenEngine” on page 377.
- Call “PECanCloseEngine” on page 282, before calling this function to make sure no print jobs are in a busy state.

VB Syntax

```
Declare Sub PECloseEngine Lib “crpe32.dll” ()
```

Delphi Syntax

```
procedure PECloseEngine
    stdcall;
```

dBASE for Windows Syntax

```
EXTERN CVOID PECloseEngine () CRPE.DLL
```

PEClosePrintJob

Use PEClosePrintJob to close the print job. If printing has not yet finished, it continues; if the preview window is open, it stays open. This function is used as a mandatory part of each Custom-Print Link to shut down the print job once it has finished printing.

C Syntax

```
BOOL CRPE_API PEClosePrintJob (
        short printJob );
```

Parameter

printJob	Specifies the print job that you want to close.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

Once this function has been called, most other Crystal Report Engine functions except “PEOpenPrintJob” on page 378; “PECloseEngine” on page 287; “PEGetVersion” on page 369; and “PELogOnServer” on page 374, cannot be called.

VB Syntax

```
Declare Function PEClosePrintJob Lib "crpe32.dll" (ByVal printJob _
        As Integer) As Integer
```

Delphi Syntax

```
function PEClosePrintJob (
    printJob: Word
    ): Bool stdcall;
```

dBASE for Windows Syntax

```
EXTERN CLOGICAL PEClosePrintJob (CWORD) CRPE.DLL
```

PECloseSubreport

Use PECloseSubreport to close the specified subreport.

C Syntax

```
BOOL CRPE_API PECloseSubreport (
        short printJob );
```

Parameter

printJob	Specifies the print job for the subreport that you want to close.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

Once this function has been called, "PEGetSubreportInfo" on page 367, or any other Crystal Report Engine function that applies to the subreport cannot be called.

VB Syntax

```
Declare Function PECloseSubreport Lib "crpe32.dll" (
    ByVal printJob As Integer ) As Integer
```

Delphi Syntax

```
function PECloseSubreport (
    printJob: Word
    ): Bool stdcall;
```

dBASE for Windows Syntax

```
EXTERN CLOGICAL PECloseSubreport(CWORD) CRPE.DLL
```

PECloseWindow

Use PECloseWindow to close the preview window. Use this function as part of a Custom-Print Link to enable the user to review the report in the preview window and then to close the window in response to a user event. Use PECloseWindow to replace the Close button when PEShowPrintControls disables the Print Control buttons.

C Syntax

```
void CRPE_API PECloseWindow (
        short printJob );
```

Parameter

printJob	Specifies the print job for which you want to close the preview window.

Returns

Void.

Remarks

The preview window will not be closed if it is busy (generating pages, reading records, etc.).

VB Syntax

```
Declare Sub PECloseWindow Lib "crpe32.dll" ( ByVal printJob As Integer )
```

Delphi Syntax

```
procedure PECloseWindow (
    printJob: Word
    )stdcall;
```

dBASE for Windows Syntax

```
EXTERN CVOID PECloseWindow (CWORD) CRPE.DLL
```

PEConvertPFInfotoVInfo

Use PEConvertPFInfotoVInfo to convert the value returned in the CurrentValue or DefaultValue member of "PEParameterFieldInfo" on page 484, to the appropriate type and place the result in "PEValueInfo" on page 516.

C Syntax

```
BOOL CRPE_API PEConvertPFInfoToVInfo(
        void FAR *value,
        short valueType,
        PEValueInfo FAR *valueInfo );
```

Parameters

value	Specifies a pointer to the current or default parameter value (returned from "PEGetNthParameterField" on page 340) to convert.
valueType	Specifies the type of parameter field from which the value came. Use one of the "Parameter Field Value Type Constants" on page 559.
valueInfo	Specifies a pointer to "PEValueInfo" on page 516, that is used to store the value in converted form.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEConvertPFInfoToVInfo Lib "crpe32.dll" (ByVal value _
    As Any, ByVal valueType As Integer, valueInfo As PEValueInfo) As
Integer
```

Delphi Syntax

```
function PEConvertPFInfoToVInfo (
    value: PChar;
    valueType: Word;
    var valueInfo: PEValueInfo
    ): Bool stdcall;
```

PEConvertVInfotoPFInfo

Use PEConvertVInfotoPFInfo to convert a value contained in "PEValueInfo" on page 516, into the binary representation expected by "PESetNthParameterField" on page 423.

C Syntax

```
BOOL CRPE_API PEConvertVInfoToPFInfo (
        PEValueInfo FAR *valueInfo,
        WORD FAR *valueType,
        void FAR *value );
```

Parameters

valueInfo	Specifies a pointer to "PEValueInfo" on page 516, that contains the value to be converted.
valueType	Specifies a pointer to the type of the value.
value	Specifies a pointer used to store the value in converted form for use in "PESetNthParameterField" on page 423.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEConvertVInfoToPFInfo Lib "crpe32.dll" (valueInfo _
    As PEValueInfo, ByVal valueType as Integer, ByVal value As Any) As
Integer
```

Delphi Syntax

```
function PEConvertVInfoToPFInfo (
    var valueInfo: PEValueInfo;
    var valueType: Word;
    value: Pchar
    ): Bool stdcall;
```

PEDeleteNthGroupSortField

Use PEDeleteNthGroupSortField to remove the specified group sort field from the sort order. This function is used as part of a Custom-Print Link whenever you want to delete group sort fields that were established for the report at design time. When you give the user the ability to delete group sort field(s) at print time, your link must include code to replace sortFieldN with user-generated values.

This function can be used by itself to delete an existing group sort field when the sort field number is already known or as one of a series of functions ("PEGetNGroupSortFields" on page 323 called once; "PEGetNthGroupSortField" on page 335 and "PEGetHandleString" on page 318 called together as many times as needed to identify the correct sort field; and PEDeleteNthGroupSortField called once, when the correct sort field is identified). The series can be used in a Custom-Print Link to identify and then delete an existing group sort field and / or sort order in response to a user selection at print time.

C Syntax

```
BOOL CRPE_API PEDeleteNthGroupSortField (
        short printJob,
        short sortFieldN );
```

Parameters

printJob	Specifies the print job from which you want to delete a group sort field.
sortFieldN	Specifies the 0-based number of the sort field that you want to delete. The first group sort field is field 0. If N = 0, the function will delete the first group sort field. If the report has N group sort fields, the function can be called with sortFieldN between 0 and N-1.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

This function should be called before "PEStartPrintJob" on page 448 or the results may be inconsistent or unexpected.

VB Syntax

```
Declare Function PEDeleteNthGroupSortField Lib "crpe32.dll" ( _
    ByVal printJob As Integer, ByVal SortFieldN As Integer ) As Integer
```

Delphi Syntax

```
function PEDeleteNthGroupSortField (
    printJob: Word;
    sortFieldN: integer
    ): Bool stdcall;
```

dBASE for Windows Syntax

```
EXTERN CLOGICAL PEDeleteNthGroupSortField (CWORD, CWORD) CRPE.DLL
```

PEDeleteNthParameterDefaultValue

Use to remove a default parameter value for the specified parameter in a report.

C Syntax

```
BOOL CRPE_API PEDeleteNthParameterDefaultValue (
        short printJob,
        const char FAR *parameterFieldName,
        const char FAR *reportName,
        short index );
```

Parameters

printJob	Specifies the print job from which you want to delete a default value.
parameterFieldName	Specifies a pointer to the string containing the parameter field name.
reportName	Specifies a pointer to a string containing the report name. See Remarks below.
index	Specifies the index number of the default value to be deleted.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

Regarding parameter reportName:

- For the main report, pass an empty string ("").
- For a subreport, pass the file path and name of the subreport as a NULL-terminated string.

VB Syntax

```
Declare Function PEDeleteNthParameterDefaultValue Lib "crpe32.dll" ( _
    ByVal printJob As Integer, ByVal parameterFieldName As String, _
    ByVal reportName As String, ByVal index As Integer ) As Integer
```

Delphi Syntax

```
function PEDeleteNthParameterDefaultValue (
    printJob: smallint;
    const parameterFieldName: PChar;
    const reportName: PChar;
    index: smallint
    ): BOOL stdcall;
```

PEDeleteNthSortField

Use PEDeleteNthSortField to remove the specified sort field from the sort order. This function is used as part of a Custom-Print Link whenever you want to delete sort fields that were established for the report at design time. When you give the user the ability to delete sort field(s) at print time, your link must include code to replace sortFieldN with user-generated values.

This function can be used by itself to delete an existing sort field when the sort field number is already known or as one of a series of functions (“PEGetNSortFields” on page 330, called once; “var reportAlertInfo : PEReportAlertInfo) : boolean stdcall;” on page 343 and “PEGetHandleString” on page 318 called together as many times as needed to identify the correct sort field; and PEDeleteNthSortField called once, when the correct sort field is identified). The series can be used in a Custom-Print Link to identify and then delete an existing sort field and/or sort order in response to a user selection at print time.

C Syntax

```
BOOL CRPE_API PEDeleteNthSortField (
        short printJob,
        short sortFieldN );
```

Parameters

printJob	Specifies the print job from which you want to delete a sort field.
sortFieldN	Specifies the 0-based number of the sort field you want to delete. The first sort field is field 0. If N = 0, the function will delete the first sort field, If the report has N sort fields, you can call the function with sortFieldN between 0 and N-1.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

This function should be called before "PEStartPrintJob" on page 448, or the results may be inconsistent or unexpected.

VB Syntax

```
Declare Function PEDeleteNthSortField Lib "crpe32.dll" ( _
    ByVal printJob As Integer, ByVal SortFieldN As Integer ) As Integer
```

Delphi Syntax

```
function PEDeleteNthSortField (
    printJob: Word;
    sortFieldN: integer
    ): Bool stdcall;
```

dBASE for Windows Syntax

```
EXTERN CLOGICAL PEDeleteNthSortField (CWORD, CWORD) CRPE.DLL
```

PEDiscardSavedData

Use PEDiscardSavedData to discard the data that was previously saved with the report. If a report has been saved with data, you can use this function to discard the saved data, forcing the Crystal Report Engine to retrieve new data the next time the report is printed.

C Syntax

```
BOOL CRPE_API PEDiscardSavedData (
        short printJob );
```

Parameter

printJob	Specifies the print job for which you want to discard the saved data.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

This function should be called before "PEStartPrintJob" on page 448, or the results may be inconsistent or unexpected.

VB Syntax

```
Declare Function PEDiscardSavedData Lib "crpe32.dll" (
    ByVal printJob As Integer ) As Integer
```

Delphi Syntax

```
function PEDiscardSavedData (
    printJob: Word
    ): Bool stdcall;
```

dBASE for Windows Syntax

```
EXTERN CLOGICAL PEDiscardSavedData (CWORD) CRPE.DLL
```

PEEnableNthAlert

Use PEEnableNthAlert to enable or disable a Report Alert.

C Syntax

```
BOOL CRPE_API PEEnableNthAlert (short printJob,
                                short alertN,
                                BOOL enabled);
```

Parameters

printJob	Specifies the print job for which you want to enable or disable Report Alerts.
alertN	Specifies the Report Alert to enable.
enabled	Specifies wether or not to enable the Report Alert. Set to TRUE to enable, or set to FALSE to disable.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEEnableNthAlert Lib "crpe32.dll" (ByVal printJob%,
ByVal alertN%, enabled As Integer) As Integer
```

Delphi Syntax

```
function PEEnableNthAlert(
   printJob : Smallint;
   alertN : Smallint;
   enabled : boolean) : boolean stdcall;
```

PEEnableEvent

Use PEEnableEvent to enable or disable print job events. All events are disabled by default.

C Syntax

```
BOOL CRPE_API PEEnableEvent (
        short printJob,
        PEEnableEventInfo Far *enableEventInfo );
```

Parameters

printJob	Specifies the print job for which you want to enable or disable events.
enableEventInfo	Specifies a pointer to “PEEnableEventInfo” on page 457.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Delphi Syntax

```
function PEEnableEvent (
printJob: Word;
Var enableEventInfo: PEEnableEventInfo
): Bool stdcall;
```

PEEnableProgressDialog

Use PEEnableProgressDialog to specify whether the Progress dialog box is enabled. The Progress dialog box displays the progress of the report when it is running (records read, records selected, etc.).

C Syntax

```
BOOL CRPE_API PEEnableProgressDialog (
        short printJob,
        BOOL enable );
```

Parameters

printJob	Specifies the print job for which you want to enable / disable the progress dialog box.
enable	Specifies whether or not the progress dialog box is enabled. If enable is set to TRUE, the progress dialog box is enabled. If it is set to FALSE, the dialog box is disabled.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

This dialog box is enabled by default.

VB Syntax

```
Declare Function PEEnableProgressDialog Lib "crpe32.dll" (ByVal printJob
_
        As Integer, ByVal enable As Integer) As Integer
```

Delphi Syntax

```
function PEEnableProgressDialog (
    printJob: Word;
    enable: Bool
    ): Bool stdcall;
```

PEExportPrintWindow

Use PEExportPrintWindow to export the report displayed in the preview window to disk file or e-mail. This function can be used in a Custom-Print Link to enable the user to preview the report in the preview window, and then, if everything looks satisfactory, to export the report to disk file or e-mail in response to a user event. You can use PEExportPrintWindow to replace the Export button when "PEShowPrintControls" on page 447, disables the Print Control buttons.

C Syntax

```
BOOL CRPE_API PEExportPrintWindow (
        short printJob,
        BOOL toMail,
        BOOL waitUntilDone );
```

Parameters

printJob	Specifies the print job that you want to export to a disk file or to e-mail.
toMail	Boolean value indicates whether or not the function is to export to e-mail.
waitUntilDone	BOOL. Reserved. This parameter must always be set to TRUE.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEExportPrintWindow Lib "crpe32.dll" (
    ByVal printJob As Integer, ByVal ToMail As Integer,
    ByVal WaitUntilDone As Integer ) As Integer
```

Delphi Syntax

```
function PEExportPrintWindow (
    printJob: Word;
    toMail: Bool;
    waitUntilDone: Bool
    ): Bool stdcall;
```

dBASE for Windows Syntax

```
EXTERN CLOGICAL PEExportPrintWindow (CWORD, CLOGICAL, CLOGICAL) CRPE.DLL
```

PEExportTo

Use PEExportTo to export a print job using the name, format, and destination specified in the "PEExportOptions" on page 458. Use this function any time you want to print a report to a file or export it for use in other programs.

C Syntax

```
BOOL CRPE_API PEExportTo (
        short printJob,
        PEExportOptions FAR *options );
```

Parameters

printJob	Specifies the print job that you want to export.
options	Specifies a pointer to "PEExportOptions" on page 458.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEExportTo Lib "crpe32.dll" ( ByVal printJob As
Integer,
    ExportOptions As PEExportOptions ) As Integer
```

Note: Visual Basic method PEExportTo will fail unless method PEGetExportOptions is called.

Delphi Syntax

```
function PEExportTo (
    printJob: Word;
    var options: PEExportOptions
    ): Bool stdcall;
```

PEFreeDevMode

Use PEFreeDevMode to return the memory associated with the specified DEVMODE Microsoft Windows structure to the heap. The DEVMODE structure must have been retrieved from "PESelectPrinter" on page 389, or "PEGetSelectedPrinter" on page 363.

C Syntax

```
BOOL CRPE_API PEFreeDevMode (
    short printJob,
    DEVMODEA FAR *mode );
```

Parameters

printJob	Specifies the print job for which you want to release the specified "DEVMODE" on page 533, Windows API structure.
mode	Specifies a pointer to the "DEVMODE" on page 533, Windows API structure that you want to release.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEFreeDevMode Lib "crpe32.dll" ( ByVal printJob As
Integer,
    mode As Any ) As Integer
```

PEGetAllowPromptDialog

Use PEGetAllowPromptDialog to determine whether prompting for parameter values is allowed for the specified job during printing.

C Syntax

```
BOOL CRPE_API PEGetAllowPromptDialog (
        short printJob );
```

Parameter

printJob	Specifies the print job for which you want to determine whether prompting for parameter values is allowed during printing.

Returns

- TRUE if prompting for parameter values is allowed.
- FALSE if prompting is not allowed.

VB Syntax

```
Declare Function PEGetAllowPromptDialog Lib "crpe32.dll" (ByVal printJob
_
        As Integer) As Integer
```

Delphi Syntax

```
function PEGetAllowPromptDialog (
    printJob: Smallint
    ): Bool stdcall;
```

PEGetAreaFormat

Use PEGetAreaFormat to retrieve the area format settings for selected areas in the specified report and supply them as member values for “PESectionOptions” on page 501. Use this function to update the area formats and pass them back using “PESetAreaFormat” on page 391.

C Syntax

```
BOOL CRPE_API PEGetAreaFormat (
    short printJob,
        short areaCode,
        PESectionOptions FAR *options );
```

Parameters

printJob	Specifies the print job from which you want to get the area format.
areaCode	Specifies the “Section Codes” on page 559, for the area for which you want to get format information. See “Working with section codes” on page 46.
options	Specifies a pointer to “PESectionOptions” on page 501, which will receive format information for the area.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEGetAreaFormat Lib “crpe32.dll” ( ByVal printJob As
Integer,
    ByVal areaCode As Integer, Options As PESectionOptions ) As Integer
```

Delphi Syntax

```
function PEGetAreaFormat (
    printJob: Word
    areaCode: integer;
    options: PESectionOptions
    ): Bool stdcall;
```

PEGetAreaFormatFormula

Use PEGetAreaFormatFormula to retrieve the text of a conditional area format formula as a string handle. Use this function in order to update the conditional area format formula and pass the changes back using “PESetAreaFormatFormula” on page 392. Use “PEGetHandleString” on page 318, to retrieve the formula text.

C Syntax

```
BOOL CRPE_API PEGetAreaFormatFormula (
        short printJob,
        short areaCode,
        short formulaName,
        HANDLE FAR *textHandle,
        short FAR *textLength );
```

Parameters

printJob	Specifies the print job from which you want to get the area format formula.
areaCode	Specifies the “Section Codes” on page 559, for the area for which you want to get format information. See “Working with section codes” on page 46.
formulaName	Specifies the name of the formatting formula for which you want to supply a new string. Use one of the PE_FFN_XXX “Area / Section Format Formula Constants” on page 541.
textHandle	Specifies a pointer to the handle of the string containing the formula text.
textLength	Specifies a pointer to the length of the formula string (in bytes) including the terminating byte.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEGetAreaFormatFormula Lib “crpe32.dll” (
    ByVal printJob As Integer, ByVal areaCode As Integer,
    ByVal formulaName As Integer, textHandle As Long,
    textLength As Integer ) As Integer
```

Delphi Syntax

```
function PEGetAreaFormatFormula (
    printJob: Word;
    areaCode: Word;
    formulaName: Word;
    var textHandle: HWnd;
    var textLength: Word
    ): Bool stdcall;
```

PEGetEnableEventInfo

Use PEGetEnableEventInfo to retrieve event enable information. Use this function to determine which events are enabled.

C Syntax

```
BOOL CRPE_API PEGetEnableEventInfo (
        short printJob,
        PEEnableEventInfo FAR *enableEventInfo );
```

Parameters

printJob	Specifies the print job for which you want to obtain information about enabled events.
enableEventInfo	Specifies a pointer to “PEEnableEventInfo” on page 457.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Delphi Syntax

```
function PEGetEnableEventInfo(
    printJob: Word;
    Var enableEventInfo: PEEnableEventInfo
    ): Bool stdcall;
```

PEGetErrorCode

Use PEGetErrorCode to retrieve a number that indicates the status of the most recent Crystal Report Engine function called. When a call to another function fails, this call gets the error code that was generated so you can take some action based on that error code. PEGetErrorCode must be called immediately after the call to the function which indicated an error.

C Syntax

```
short CRPE_API PEGetErrorCode (
        short printJob );
```

Parameter

printJob	Specifies the print job from which you want to retrieve an error code. If the most recent function was called while no print job was open, use 0 for this parameter.

Returns

- Returns the “Error Codes” on page 545, of the most recent error for the given job if the function was unsuccessful.
- Returns 0 if the function completed without error.

VB Syntax

```
Declare Function PEGetErrorCode Lib “crpe32.dll” (ByVal printJob As
Integer) _
        As Integer
```

Delphi Syntax

```
function PEGetErrorCode (
    printJob: Word
    ): Smallint stdcall;
```

dBASE for Windows Syntax

```
EXTERN CWORD PEGetErrorCode (CWORD) CRPE.DLL
```

PEGetErrorText

Use PEGetErrorText to return a string handle describing the status of the most recent Crystal Report Engine function called. This function is used with "PEGetHandleString" on page 318. These functions can be used in a Custom-Print Link to display the error string as part of an error message.

C Syntax

```
BOOL CRPE_API PEGetErrorText (
        short printJob,
        HANDLE FAR *textHandle,
        short Far *textLength );
```

Parameters

printJob	Specifies the print job from which you want to retrieve an error string. If the most recent function was called while no print job was open, use 0 this parameter.
textHandle	Specifies a pointer to the handle of the string containing the error text.
textLength	Specifies a pointer to the length of the error string (in bytes) including the terminating byte.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEGetErrorText Lib "crpe32.dll" (ByVal printJob As
Integer, _
        TextHandle As Long, TextLength As Integer) As Integer
```

Delphi Syntax

```
function PEGetErrorText (
    printJob: Word;
    var textHandle: HWnd;
    var textLength: Word
    ): Bool stdcall;
```

PEGetExportOptions

Use PEGetExportOptions to retrieve export options from the user before exporting the report. PEGetExportOptions can be used to present a series of dialog boxes that retrieve export options from your users. These options are used by the Crystal Report Engine to fill in "PEExportOptions" on page 458. Function "PEExportTo" on page 299, can then be used to set the print job destination using the information in "PEExportOptions" on page 458.

C Syntax

```
BOOL CRPE_API PEGetExportOptions (
        short printJob,
        PEExportOptions FAR *options );
```

Parameters

printJob	Specifies the print job from which you want to get export options.
options	Specifies a pointer to "PEExportOptions" on page 458.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEGetExportOptions Lib "crpe32.dll" (
    ByVal printJob As Integer, ExportOptions As PEExportOptions ) As
Integer
```

Delphi Syntax

```
function PEGetExportOptions (
    printJob: Word;
    var options: PEExportOptions
    ): Bool stdcall;
```

PEGetFieldMappingType

Use PEGetFieldMappingType to retrieve the field mapping type code for the specified report.

C Syntax

```
BOOL CRPE_API PEGetFieldMappingType (
        short printJob,
        WORD FAR *mappingType );
```

Parameters

printJob	Specifies the print job from which the field mapping type code will be retrieved.
mappingType	Specifies a pointer to the appropriate PE_FM_XXX "Field Mapping Type Constants" on page 551.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

- Unmapped report fields will be removed.
- You need to activate the PE_MAPPING_FIELD_EVENT and define a callback function.

VB Syntax

```
Declare Function PEGetFieldMappingType Lib "crpe32.dll" (
    ByVal printJob As Integer, mappingType As Integer ) As Integer
```

Delphi Syntax

```
function PEGetFieldMappingType (
    printJob: smallint;
    var mappingType: Word
    ): BOOL stdcall;
```

PEGetFormula

Use PEGetFormula to retrieve the text of the named formula as a string handle. This function is used with "PEGetHandleString" on page 318. Use "PESetFormula" on page 405, to pass a formula back. The series can be used in a Custom-Print Link to identify and then change an existing formula in response to a user selection at print time.

C Syntax

```
BOOL CRPE_API PEGetFormula (
        short printJob,
        const char *formulaName,
        HANDLE FAR *textHandle,
        short FAR *textLength );
```

Parameters

printJob	Specifies the print job from which you want to retrieve the formula string.
formulaName	Specifies a pointer to the null-terminated string that contains the name of the formula for which you want to retrieve the formula string.
textHandle	Specifies a pointer to the handle of the string containing the formula text.
textLength	Specifies a pointer to the length of the formula string (in bytes) including the terminating byte.

Returns

- TRUE if the call is successful.
- FALSE if the named formula does not exist in the report.

VB Syntax

```
Declare Function PEGetFormula Lib "crpe32.dll" (ByVal printJob As
Integer, ByVal FormulaName As String, TextHandle As Long, TextLength As
Integer) As Integer
```

Delphi Syntax

```
function PEGetFormula (
    printJob: Word;
    formulaName: PChar;
    var textHandle: HWnd;
    var textLength: Word
    ): Bool stdcall;
```

PEGetFormulaSyntax

Use PEGetFormulaSyntax to retrieve the syntax information associated with the formula addressed in the last formula API call.

C Syntax

```
BOOL CRPE_API PEGetFormulaSyntax (
    short printJob,
    PEFormulaSyntax FAR *formulaSyntax );
```

Parameters

printJob	Specifies the print job for which you want to determine formula syntax.
formulaSyntax	Specifies a pointer to "PEFormulaSyntax" on page 466, which will contain the information that you want to retrieve. See Remarks below.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

- If PEGetFormulaSyntax is called before any Formula API is called, then the default value returned will be PE_UNCHANGED.
- For running total condition formula:
  - formulaSyntax[0] is the syntax for the evalFormula.
  - formulaSyntax[1] is the syntax for the reset formula.

VB Syntax

```
Declare Function PEGetFormulaSyntax Lib "crpe32.dll" (
    ByVal printJob As Integer, formulaSyntax As PEFormulaSyntax ) As
Integer
```

PEGetGraphAxisInfo

Use PEGetGraphAxisInfo to retrieve the several chart axis options that are available.

C Syntax

```
BOOL CRPE_API PEGetGraphAxisInfo (
        short printJob,
        short sectionN,
        short graphN,
        PEGraphAxisInfo FAR * graphAxisInfo );
```

Parameters

printJob	Specifies the print job from which you want to retrieve chart axis information.
sectionN	Specifies the section of the report containing the chart for which you want to retrieve chart axis information.
graphN	Specifies for which chart within the section you want to retrieve the chart axis information. This value is 0-based. Charts are numbered based on their order of insertion into the report.
graphAxisInfo	Specifies a pointer to “PEGraphAxisInfo” on page 468, which will contain the information retrieved.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEGetGraphAxisInfo Lib "crpe32.dll" (
    ByVal printJob As Integer, ByVal sectionN As Integer,
    ByVal graphN As Integer, graphAxisInfo As PEGraphAxisInfo ) As
Integer
```

Delphi Syntax

```
function PEGetGraphAxisInfo (
  printJob          : Smallint;
  sectionN          : Smallint;
  graphN            : Smallint;
  var graphAxisInfo : PEGraphAxisInfo): Bool; {$ifdef WIN32} stdcall;
{$endif}
```

PEGetGraphFontInfo

Use PEGetGraphFontInfo to retrieve the font information set for the specified chart.

C Syntax

```
BOOL CRPE_API PEGetGraphFontInfo (
        short printJob,
        short sectionN,
        short graphN,
        WORD titleFontType,
        PEFontColorInfo FAR * fontColourInfo );
```

Parameters

printJob	Specifies the print job for which you want to retrieve chart font information.
sectionN	Specifies the number of the section in which the chart appears. This parameter should be within the range obtained by “PEGetNSections” on page 328.
graphN	Specifies for which chart within the section you want to retrieve the font information. This value is 0-based. Charts are numbered based on their order of insertion into the report.
titleFontType	Uses one of the PE_GTF_XXX “Graph Text Font Constants” on page 554.
fontColourInfo	Specifies a pointer to PEFontColorInfo, which will contain the information retrieved.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEGetGraphFontInfo Lib "crpe32.dll" (
    ByVal printJob As Integer, ByVal sectionN As Integer,
```

```
    ByVal graphN As Integer, ByVal titleFontType As Integer,
    fontColourInfo As PEFontColorInfo ) As Integer
```

Delphi Syntax

```
function PEGetGraphFontInfo (
  printJob           : Smallint;
  sectionN           : Smallint;
  graphN             : Smallint;
  titleFontType      : Word;
  var fontColourInfo : PEFontColorInfo): Bool; {$ifdef WIN32}
stdcall;{$endif}
```

PEGetGraphOptionInfo

Use PEGetGraphOptionInfo to retrieve display options set for the specified chart.

C Syntax

```
BOOL CRPE_API PEGetGraphOptionInfo (
        short printJob,
        short sectionN,
        short graphN,
        PEGraphOptionInfo FAR * graphOptionInfo );
```

Parameters

printJob	Specifies the print job for which you want to retrieve chart display information.
sectionN	Specifies the number of the section in which the chart appears. This parameter should be within the range obtained by “PEGetNSections” on page 328.
graphN	Specifies for which chart within the Charts are numbered based on their order of insertion into the report.
graphOption Info	Specifies a pointer to “PEGraphOptionInfo”, which will contain the retrieved information.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEGetGraphOptionInfo Lib "crpe32.dll" (
    ByVal printJob As Integer, ByVal sectionN As Integer,
    ByVal graphN As Integer, graphOptionInfo As PEGraphOptionInfo ) As
Integer
```

Delphi Syntax

```
function PEGetGraphOptionInfo (
  printJob           : Smallint;
```

```
sectionN            : Smallint;
graphN              : Smallint;
var graphOptionInfo : PEGraphOptionInfo): Bool;
      {$ifdef WIN32} stdcall; {$endif}
```

PEGetGraphTextDefaultOption

Use PEGetGraphTextDefaultOption to determine whether or not default chart titles will be displayed.

C Syntax

```
BOOL CRPE_API PEGetGraphTextDefaultOption (
    short printJob,
    short sectionN,
    short graphN,
    WORD titleType,
    BOOL FAR *useDefault );
```

Parameters

printJob	Specifies the print job for which you want to retrieve chart title default option information.
sectionN	Specifies the 0-based index number of the section in which the chart appears. This parameter should be within the range obtained by "PEGetNSections" on page 328.
graphN	Specifies the 0-based index number of the chart for which you want to retrieve the chart title default option information. Charts are numbered based on their order of insertion into the report.
titleType	Specifies the title type. Use one of the PE_GTT_XXX "Graph Title Type Constants" on page 555.
useDefault	Specifies a pointer to the Boolean value indicating whether or not to default chart title will be displayed.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEGetGraphTextDefaultOption Lib "crpe32.dll" (
    ByVal printJob As Integer, ByVal sectionN As Integer,
    ByVal graphN As Integer, ByVal titleType As Integer,
    useDefault As Long ) As Integer
```

PEGetGraphTextInfo

Use PEGetGraphTextInfo to retrieve the title text information set for the specified chart. This function is used with "PEGetHandleString" on page 318.

C Syntax

```
BOOL CRPE_API PEGetGraphTextInfo (
        short printJob,
        short sectionN,
        short graphN,
        WORD titleType,
        HANDLE FAR *title,
        short FAR *titleLength );
```

Parameters

printJob	Specifies the print job for which you want to retrieve title text information.
sectionN	Specifies the number of the section in which the chart appears. This parameter should be within the range obtained by "PEGetNSections" on page 328.
graphN	Specifies the 0-based index number of the chart for which you want to retrieve the title text information. Charts are numbered based on their order of insertion into the report.
titleType	Specifies the title type. Use one of the PE_GTT_XXX "Graph Title Type Constants" on page 555.
title	Specifies a pointer to the handle of the title.
titleLength	Specifies a pointer to the length of the title.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEGetGraphTextInfo Lib "crpe32.dll" (
    ByVal printJob As Integer, ByVal sectionN As Integer,
    ByVal graphN As Integer, ByVal titleType As Integer,
    title As Long, titleLength As Integer ) As Integer
```

Delphi Syntax

```
function PEGetGraphTextInfo (
  printJob        : Smallint;
  sectionN        : Smallint;
  graphN          : Smallint;
  titleType       : Word;
  var title       : Hwnd;
  var titleLength : Smallint): Bool; {$ifdef WIN32} stdcall; {$endif}
```

PEGetGraphTypeInfo

Use PEGetGraphTypeInfo to retrieve information about the type of the specified chart.

C Syntax

```
BOOL CRPE_API PEGetGraphTypeInfo (
        short printJob,
        short sectionN,
        short graphN,
        PEGraphTypeInfo FAR * graphTypeInfo );
```

Parameters

printJob	Specifies the print job for which you want to retrieve chart type information.
sectionN	Specifies the number of the section in which the chart appears. This parameter should be within the range obtained by "PEGetNSections" on page 328.
graphN	Specifies for which chart within the section you want to retrieve the type. This value is 0-based. Charts are numbered based on their order of insertion into the report.
graphTypeInfo	Specifies a pointer to "PEGraphTypeInfo" on page 473, which will contain the retrieved information.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEGetGraphTypeInfo Lib "crpe32.dll" (
    ByVal printJob As Integer, ByVal sectionN As Integer,
    ByVal graphN As Integer, graphTypeInfo As PEGraphTypeInfo ) As
Integer
```

Delphi Syntax

```
function PEGetGraphTypeInfo (
  printJob          : Smallint;
  sectionN          : Smallint;
  graphN            : Smallint;
  var graphTypeInfo : PEGraphTypeInfo): Bool; {$ifdef WIN32} stdcall;
{$endif}
```

PEGetGroupCondition

Use PEGetGroupCondition to determine the group condition information for a selected group section in the specified report. Use this function to retrieve the group condition for a group section and use "PESetGroupCondition" on page 413, to change the group condition once it is known.

C Syntax

```
BOOL CRPE_API PEGetGroupCondition (
        short printJob,
        short sectionCode,
        HANDLE FAR *conditionFieldHandle,
        short FAR *conditionFieldLength,
        short FAR *condition,
        short FAR *sortDirection );
```

Parameters

printJob	Specifies the print job that you want to query to determine the group conditions for a selected group.
sectionCode	Specifies the "Section Codes" on page 559, for the report section for which you want to get the group condition. See "Working with section codes" on page 46.
conditionField Handle	Specifies a pointer to the handle of the condition field for the selected group section.
conditionField Length	Specifies a pointer to the length of the condition field for the selected group section.
condition	Specifies a pointer to the condition setting for the selected group section. See Remarks below.
sortDirection	Specifies a pointer to the sort direction setting for the selected group section. Uses one of the PE_SF_XXX "Sort Order Constants" on page 560.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

The *condition* parameter returns a value that encodes both the condition and the type of the condition field. You need to apply a condition mask (PE_GC_CONDITIONMASK) or a type mask (PE_GC_TYPEMASK) against this value using a bitwise AND to determine the condition or type respectively. For example:

```
short result;
result = *condition & PE_GC_TYPEMASK;
if (result == PE_GC_TYPEDATE)
{
    //what you want it to do
}
```

Type can be any of the following PE_GC_TYPEXXX constants:

- PE_GC_TYPEOTHER -- Any data type other than date or Boolean.
- PE_GC_TYPEDATE -- Date data type.
- PE_GC_TYPEBOOLEAN -- Boolean data type.
- PE_GC_TYPETIME -- Time data type

VB Syntax

```
Declare Function PEGetGroupCondition Lib "crpe32.dll" (ByVal printJob As
Integer, ByVal sectionCode As Integer, ConditionFieldHandle As Long,
ConditionFieldLength As Integer, Condition As Integer, SortDirection As
Integer) As Integer
```

Delphi Syntax

```
function PEGetGroupCondition (
    printJob: Word;
    sectionCode: integer;
    var conditionFieldHandle: Hwnd;
    var conditionFieldLength: Word;
    var condition: Word;
    var sortDirection: Word
    ): Bool stdcall;
```

PEGetGroupOptions

Use PEGetGroupOptions to retrieve the current settings for specified groups in your report.

C Syntax

```
BOOL CRPE_API PEGetGroupOptions (
        short printJob,
        short groupN,
        PEGroupOptions FAR *groupOptions );
```

Parameters

printJob	Specifies the print job for which you want to obtain information about the group options settings.
groupN	Specifies the 0-based group number.
groupOptions	Specifies a pointer to "PEGroupOptions" on page 474.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

This function gets all the information retrieved with “PEGetGroupCondition” on page 315, plus additional group options such as repeat group header, keep group together, and top / bottom n group sorting.

VB Syntax

```
Declare Function PEGetGroupOptions Lib “crpe32.dll” (ByVal printJob As
Integer, ByVal groupN As Integer, groupOptions As PEGroupOptions) As
Integer
```

Delphi Syntax

```
function PEGetGroupOptions
    printJob: Word;
    groupN: smallint;
    var groupOptions: PEGroupOptions
    ): integer stdcall;
```

PEGetGroupSelectionFormula

Use PEGetGroupSelectionFormula to retrieve the string handle for the group selection formula used in the specified report. This function is used with “PEGetHandleString” on page 318. Use “PESetGroupSelectionFormula” on page 415, to pass back a formula. This series can be used in a Custom-Print Link to identify and then change an existing group selection formula in response to a user selection at print time.

C Syntax

```
BOOL CRPE_API PEGetGroupSelectionFormula (
        short printJob,
        HANDLE FAR *textHandle,
        short FAR *textLength );
```

Parameters

printJob	Specifies the print job for which you want to retrieve the group selection formula string.
textHandle	Specifies a pointer to the handle of the string containing the formula text.
textLength	Specifies a pointer to the length of the formula string (in bytes) including the terminating byte.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEGetGroupSelectionFormula Lib "crpe32.dll" (ByVal
printJob As Integer, TextHandle As Long, TextLength As Integer) As
Integer
```

Delphi Syntax

```
function PEGetGroupSelectionFormula (
    printJob: Word;
    var textHandle: HWnd;
    var textLength: Word
    ): Bool stdcall;
```

PEGetHandleString

Use PEGetHandleString to retrieve the text to which the string handle is pointing. The buffer will obtain the actual text. This function is used in conjunction with functions that return variable length strings. After your program allocates a buffer of sufficient size, this function moves the string from the string handle to the buffer.

C Syntax

```
BOOL CRPE_API PEGetHandleString (
        HANDLE textHandle,
        char FAR *buffer,
        short bufferLength );
```

Parameters

textHandle	Specifies the handle of the string containing the text of interest. This handle is obtained from a variable length string function.
buffer	Specifies a pointer to the buffer into which you want the string copied.
bufferLength	Specifies the length of the buffer in bytes, including the terminating null byte. This value should be identical to the length of the string obtained by the variable length string function.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

- When you call the function that produces the string, it returns a length that includes a provision for the null byte at the end of the string. A buffer set to that length will hold the entire string including the terminating null byte.
- PEGetHandleString will copy, at most, the number of bytes indicated by bufferLength, ensuring that the string in the buffer is NULL-terminated.

- You can only use this call once with a given string handle because the string handle is discarded once the function is called. If you expect to use the string later, you will need to save it.

VB Syntax

```
Declare Function PEGetHandleString Lib "crpe32.dll" (ByVal textHandle _
   As Long,  ByVal Buffer As String, ByVal BufferLength As Integer) As
Integer
```

Delphi Syntax

```
function PEGetHandleString (
    textHandle: HWnd;
    buffer: PChar;
    bufferLength: integer
    ): Bool stdcall;
```

PEGetJobStatus

Use PEGetJobStatus to determine the status of a print job. You can use this function in a number of programming situations, for example:

- to trigger error messages, such as when a print job fails (due to insufficient memory, insufficient disk space, etc.);
- to trigger screen displays (hourglass, series of graphics, etc.) that confirm to the user that work is in progress; or
- to find out whether a job was cancelled by the user after “PEStartPrintJob” on page 448, returns.

C Syntax

```
short CRPE_API PEGetJobStatus (
        short printJob,
        PEJobInfo FAR *jobInfo );
```

Parameters

printJob	Specifies the print job for which you want to determine printing status.
jobInfo	Specifies a pointer to “PEJobInfo” on page 478, which will contain the information this function retrieves.

Returns

- Returns 0 if PEOpenEngine or PEOpenPrintJob has not been called successfully.
- Otherwise, returns the “Job Status Constants” on page 558 for the specified print job.

VB Syntax

```
Function PEGetJobStatus(ByVal job As Integer, info As PEJobInfo) As
Integer
' To work around the problem of 4 - byte alignment the PEGetJobStatus
' call has been re-declared here. When your application calls
PEGetJobStatus
' it is calling this function which in turn calls CRPE32.DLL.
```

Delphi Syntax

```
function PEGetJobStatus (
    printJob: Word;
    var jobInfo: PEJobInfo
    ): smallint stdcall;
```

PEGetMargins

Use PEGetMargins to retrieve the page margin settings for the specified report. Use this function to retrieve the report margins and “PESetMargins” on page 416, to set the report margins.

C Syntax

```
BOOL CRPE_API PEGetMargins (
        short printJob,
        short FAR *left,
        short FAR *right,
        short FAR *top,
        short FAR *bottom );
```

Parameters

printJob	Specifies the print job that you want to query to retrieve margin information.
left	Specifies a pointer to the value of the left margin.
right	Specifies a pointer to the value of the right margin.
top	Specifies a pointer to the value of the top margin.
bottom	Specifies a pointer to the value of the bottom margin.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEGetMargins Lib “crpe32.dll” ( ByVal printJob As
Integer,
    LeftMargin As Integer, RightMargin As Integer,
    TopMargin As Integer, BottomMargin As Integer ) As Integer
```

Delphi Syntax

```
function PEGetMargins (
    printJob: Word;
    var left: Word;
    var right: Word;
    var top: Word;
    var bottom: Word
    ): Bool stdcall;
```

PEGetNDetailCopies

Use PEGetNDetailCopies to retrieve the number of copies of each Details section in the report that are to be printed. Use this function to find out how many times each Details section of the report will be printed. To change the number of times each Details section is printed, use “PESetNDetailCopies” on page 417.

C Syntax

```
BOOL CRPE_API PEGetNDetailCopies (
        short printJob,
        short FAR *nDetailCopies );
```

Parameters

printJob	Specifies the print job that you want to query to determine how many times each Details section is to be printed.
nDetailCopies	Specifies a pointer to the number of copies of the Details section to be printed.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEGetNDetailCopies Lib “crpe32.dll” (
    ByVal printJob As Integer, nDetailCopies As Integer ) As Integer
```

Delphi Syntax

```
function PEGetNDetailCopies(
    printJob: Word;
    var nDetailCopies: integer
    ): Bool stdcall;
```

PEGetNFormulas

Use PEGetNFormulas to determine the number of formulas in the specified report. To retrieve the formula by number, use "PEGetNthFormula" on page 334.

C Syntax

```
short CRPE_API PEGetNFormulas (
        short printJob );
```

Parameter

printJob	Specifies the print job that you want to query to determine the number of formulas it contains.

Returns

- Returns the number of formulas in the report.
- Returns -1 if an error occurs.

VB Syntax

```
Declare Function PEGetNFormulas Lib "crpe32.dll" (ByVal printJob As
Integer) As Integer
```

Delphi Syntax

```
function PEGetNFormulas(
    printJob: Word
    ): Smallint stdcall;
```

dBASE for Windows Syntax

```
EXTERN CWORD PEGetNFormulas (CWORD) CRPE.DLL
```

PEGetNGroups

Use PEGetNGroups to determine the number of groups in the specified report.

C Syntax

```
short CRPE_API PEGetNGroups (
        short printJob );
```

Parameter

printJob	Specifies the print job that you want to query to determine the number of groups.

Returns

- Returns the number of groups in the report.
- Returns -1 if an error occurs.

VB Syntax

```
Declare Function PEGetNGroups Lib "crpe32.dll" (ByVal printJob As
Integer) As Integer
```

Delphi Syntax

```
function PEGetNGroups (
    printJob: Word
    ): smallint stdcall;
```

dBASE for Windows Syntax

```
EXTERN CWORD PEGetNGroups (CWORD) CRPE.DLL
```

PEGetNGroupSortFields

Use PEGetNGroupSortFields to retrieve the number of group sort fields in the specified report. This function is typically used as one of a series of functions (PEGetNGroupSortFields called once; "PEGetNthGroupSortField" on page 335 and "PEGetHandleString" on page 318 called as many times as needed to identify the correct group sort field; and "PESetNthAlertConditionFormula" on page 418 called once, when the correct group sort field is identified. The series can be used in a Custom-Print Link to identify and then change an existing group sort field and / or sort order in response to a user selection at print time.

C Syntax

```
short CRPE_API PEGetNGroupSortFields (
        short printJob );
```

Parameter

printJob	Specifies the print job for which you want to determine the number of contained sort fields.

Returns

- Returns the number of group sort fields.
- Returns 0 if there are no group sort fields defined.
- Returns -1 if an error occurs.

VB Syntax

```
Declare Function PEGetNGroupSortFields Lib "crpe32.dll" ( _
    ByVal printJob As Integer ) As Integer
```

Delphi Syntax

```
function PEGetNGroupSortFields (
    printJob: Word
    ): Smallint stdcall;
```

dBASE for Windows Syntax

```
EXTERN CWORD PEGetNGroupSortFields (CWORD) CRPE.DLL
```

PEGetNPages

Use PEGetNPages to retrieve the number of pages in the report. This information can be used to allow the user to display a specific page in a preview window using PEShowNthPage ("PEShow...Page" on page 446), for example.

C Syntax

```
short CRPE_API PEGetNPages (
        short printJob );
```

Parameter

printJob	Specifies the print job for which you want to determine the number of pages.

Returns

- Returns the number of pages in the report if the call is successful.
- Returns -1 if an error occurs

VB Syntax

```
Declare Function PEGetNPages Lib "crpe32.dll" (ByVal printJob As
Integer) As Integer
```

Delphi Syntax

```
function PEGetNPages (
    printJob: Word
    ): Smallint stdcall;
```

PEGetNParameterCurrentRanges

Use PEGetNParameterCurrentRanges to retrieve the number of value ranges currently associated with the specified parameter field in a report. See "Working with Parameter Values and Ranges" on page 45.

C Syntax

```
short CRPE_API PEGetNParameterCurrentRanges (
        short printJob,
        const char FAR *parameterFieldName,
        const char FAR *reportName );
```

Parameters

printJob	Specifies the print job for which you want to retrieve parameter current range information.
parameterFieldName	Specifies a pointer to the string containing the parameter field name.
reportName	Specifies a pointer to the string containing the report name. See Remarks below.

Returns

- Returns the number of value ranges associated with the specified parameter field.
- Returns -1 if an error occurs.

VB Syntax

```
Declare Function PEGetNParameterCurrentRanges Lib "crpe32.dll" ( _
    ByVal printJob As Integer, ByVal parameterFieldName As String, _
    ByVal reportName As String ) As Integer
```

Delphi Syntax

```
procedure PEGetNParameterCurrentRanges (
    printJob: smallint;
    const parameterFieldName: PChar;
    const reportName: PChar;
    ): WORD stdcall;
```

PEGetNParameterCurrentValues

Use PEGetNParameterCurrentValues to determine the number of values currently stored in the specified parameter field of a report. See "Working with Parameter Values and Ranges" on page 45.

C Syntax

```
short CRPE_API PEGetNParameterCurrentValues (
        short printJob,
        const char FAR *parameterFieldName,
        const char FAR *reportName );
```

Parameters

printJob	Specifies the print job for which you want to determine the number of parameter current values.
parameterFieldName	Specifies a pointer to the string containing the parameter field name.
reportName	Specifies a pointer to the string containing the report name. See Remarks below.

Returns

- Returns the number of values currently stored in the parameter field.
- Returns -1 if an error occurs.

Remarks

Regarding parameter reportName:

- For the main report, pass an empty string ("").
- For a subreport, pass the file path and name of the subreport as a NULL-terminated string.

VB Syntax

```
Declare Function PEGetNParameterCurrentValues Lib "crpe32.dll" ( _
    ByVal printJob As Integer, ByVal parameterFieldName As String, _
    ByVal reportName As String ) As Integer
```

Delphi Syntax

```
function PEGetNParameterCurrentValues
    printJob: smallint;
    const parameterFieldName: PChar;
    const reportName: PChar
    ): Word stdcall;
```

PEGetNParameterDefaultValues

Use PEGetNParameterDefaultValues to determine the number of default values associated with the specified parameter in a report.

C Syntax

```
short CRPE_API PEGetNParameterDefaultValues (
        short printJob
        const char FAR *parameterFieldName
        const char FAR *reportName );
```

Parameters

printJob	Specifies the print job for which you want to determine the number of parameter default values.
parameterField Name	Specifies a pointer to the string containing the name of the parameter field.
reportName	Specifies a pointer to the string containing the name of the report. See Remarks below.

Returns

- Returns the number of parameter default values.
- Returns -1 if an error occurs.

Remarks

Regarding parameter reportName:

- For the main report, pass an empty string ("").
- For a subreport, pass the file path and name of the subreport as a NULL-terminated string.

VB Syntax

```
Declare Function PEGetNParameterDefaultValues Lib "crpe32.dll" ( _
    ByVal printJob As Integer, ByVal parameterFieldName As String, _
    ByVal reportName As String ) As Integer
```

Delphi Syntax

```
function PEGetNParameterDefaultValues (
    printJob: smallint;
    const parameterFieldName: PChar;
    const reportName: PChar
    ): Smallint stdcall;
```

PEGetNParameterFields

Use PEGetNParameterFields to determine the number of parameter fields used in a report, including the parameter fields in all subreports.

C Syntax

```
short CRPE_API PEGetNParameterFields (
        short printJob );
```

Parameter

printJob	Specifies the print job from which you want to retrieve a parameter field count.

Returns

- The number of parameter fields in the report.
- Returns -1 if an error occurs.

VB Syntax

```
Declare Function PEGetNParameterFields Lib "crpe32.dll" (ByVal printJob
As Integer) As Integer
```

Delphi Syntax

```
function PEGetNParameterFields (
    printJob: Word
    ): Smallint stdcall;
```

PEGetNReportAlerts

Use PEGetNReportAlerts to get the number of Report Alerts in the report associated with the job handle.

C Syntax

```
short CRPE_API PEGetNReportAlerts (short printJob);
```

Parameters

printJob	Specifies the print job from which you want to get the number of Report Alerts.

Returns

- The number of Report Alerts if the call is successful.
- -1 if the call fails.

VB Syntax

```
Declare Function PEGetNReportAlerts Lib "crpe32.dll" (
    ByVal printJob As Integer) As Integer
```

Delphi Syntax

```
ffunction PEGetNReportAlerts (printJob : Smallint) : Smallint stdcall;
```

PEGetNSections

Use PEGetNSections to retrieve the number of sections in the specified report. By default, each report has five areas, each containing one section (Report Header, Page Header, Details, Report Footer, and Page Footer). Thus, if this function were applied to a default report, five (5) would be returned. As you add groups to your report or you add sections to one or more areas, the number of sections in the report increases.

C Syntax

```
short CRPE_API PEGetNSections (
        short printJob );
```

Parameter

printJob	Specifies the print job from which you want to get a section count.

Returns

- Returns the number of sections in the report.
- Returns -1 if an error occurs.

VB Syntax

```
Declare Function PEGetNSections Lib “crpe32.dll” (
    ByVal printJob As Integer ) As Integer
```

Delphi Syntax

```
function PEGetNSections (
    printJob: Word
    ): Smallint stdcall;
```

PEGetNSectionsInArea

Use PEGetNSectionsInArea to retrieve the number of sections contained in the specified area of the specified report.

C Syntax

```
short CRPE_API PEGetNSectionsInArea (
        short printJob,
        short areaCode );
```

Parameters

printJob	Specifies the print job for which you want to determine the number of sections in the specified area.
areaCode	Specifies the area for which you want to retrieve the section count.

Returns

- Returns the number of sections in the specified area of the specified report if the call is successful.
- Returns -1 if the call fails.

VB Syntax

```
Declare Function PEGetNSectionsInArea Lib "crpe32.dll" (
    ByVal printJob As Integer, ByVal areaCode As Integer ) As Integer
```

PEGetNSortFields

Use PEGetNSortFields to retrieve the number of sort fields in the specified report. This function is typically used as one of a series of functions (PEGetNSortFields called once; “var reportAlertInfo : PEReportAlertInfo) : boolean stdcall;” on page 343 and “PEGetHandleString” on page 318 called together as many times as needed to identify the correct sort field; and “PESetNthSortField” on page 425 called once when the correct sort field is identified). The series can be used in a Custom-Print Link to identify and then change an existing sort field and / or sort order in response to a user selection at print time.

C Syntax

```
short CRPE_API PEGetNSortFields (
        short printJob );
```

Parameter

printJob	Specifies the print job for which you want to determine the number of sort fields contained.

Returns

- Returns the number of sort fields.
- Returns 0 if there are no sort fields defined.
- Returns -1 if an error occurs.

VB Syntax

```
Declare Function PEGetNSortFields Lib “crpe32.dll” ( _
    ByVal printJob As Integer ) As Integer
```

Delphi Syntax

```
function PEGetNSortFields (
    printJob: Word
    ): smallint stdcall;
```

dBASE for Windows Syntax

```
EXTERN CWORD PEGetNSortFields (CWORD) CRPE.DLL
```

PEGetNSQLExpressions

Use PEGetNSQLExpressions to retrieve the number of SQL expressions in the specified report.

C Syntax

```
short CRPE_API PEGetNSQLExpressions (
        short printJob );
```

Parameter

print job	Specifies the print job for which you want to determine the number of SQL expressions.

Returns

- Returns the number of SQL expressions in the report.
- Returns -1 if an error occurs.

VB Syntax

```
Declare Function PEGetNSQLExpressions Lib "crpe32.dll" (ByVal printJob
As Integer) As Integer
```

Delphi Syntax

```
function PEGetNSQLExpressions (
    printJob: smallint
    ): smallint stdcall;
```

PEGetNSubreportsInSection

Use PEGetNSubreportsInSection to determines the number of subreports in the specified section.

C Syntax

```
short CRPE_API PEGetNSubreportsInSection (
        short printJob,
        short sectionCode );
```

Parameters

print job	Specifies the primary report from which you want to retrieve information about the number of subreports in a section.
sectionCode	Specifies the “Section Codes” on page 559, of the section for which you want a subreport count. See “Working with section codes” on page 46.

Returns

- Returns the number of subreports in the specified section.
- Returns -1 if an error occurs.

Remarks

sectionCode can be retrieved using “PEGetSectionCode” on page 360.

VB Syntax

```
Declare Function PEGetNSubreportsInSection Lib “crpe32.dll” (
    ByVal printJob As Integer, ByVal sectionCode As Integer ) As Integer
```

Delphi Syntax

```
function PEGetNSubreportsInSection (
    printJob: Word;
    sectionCode: Smallint
    ): Smallint stdcall;
```

PEGetNTables

Use PEGetNTables to retrieve the number of tables in the open report. It counts both PC and SQL databases. This function is one of a series of functions that enable you to retrieve and update database information in an opened report so that the report can be printed using different server, database, user, and/or table location settings.

C Syntax

```
short CRPE_API PEGetNTables (
        short printJob );
```

Parameter

printJob	Specifies the print job from which you want to retrieve a table count.

Returns

- Returns the number of tables used in the report (1 = 1 table, 2 = 2 tables, etc.).
- Returns -1 if an error occurs.

Remarks

- This function can be used with all compatible PC databases (for example, Paradox, Xbase) as well as SQL databases (for example, SQL Server, Oracle, Netware).
- PEGetNTables must be called after PEOpenPrintJob and before PEStartPrintJob.

VB Syntax

```
Declare Function PEGetNTables Lib "crpe32.dll" ( ByVal printJob As
Integer _
    ) As Integer
```

Delphi Syntax

```
function PEGetNTables (
    printJob: Word
    ): smallint stdcall;
```

dBASE for Windows Syntax

```
EXTERN CWORD PEGetNTables (CWORD) CRPE.DLL
```

PEGetNthAlertInstanceInfo

Use PEGetNthAlertInstanceInfo to retrieve the PEAlertInstanceInfo structure associated with the specified Report Alert instance. The PEAlertInstanceInfo structure contains information on a specified instance of a Report Alert.

C Syntax

```
BOOL CRPE_API PEGetNthAlertInstanceInfo (short printJob,
                                         short alertN,
                                         DWORD instanceN,
                                         PEAlertInstanceInfo FAR *
                                         alertInstanceInfo);
```

Parameters

printJob	Specifies the print job from which you want to get the number of Report Alerts.
alertN	Specifies the Report Alert from which you want to get the alert instance information.
instanceN	Specifies the instance of the Report Alert from which you want to get the alert instance information.
alertInstanceInfo	Specifies a pointer to "PEAlertInstanceInfo"

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

Currently there is no function to get the number of Report Alert instances created when a report is alert is triggered. This functionality will be implemented in a future version.

In the present build only the first instance of the Report Alert is created at runtime. This limitation will be addressed in a future release.

VB Syntax

```
Declare Function PEGetNthAlertInstanceInfo Lib "crpe32.dll" (ByVal
printJob%, ByVal alertN%, ByVal instanceN As Long, alertInstanceInfo As
PEAlertInstanceInfo) As Integer
```

Delphi Syntax

```
function PEGetNthAlertInstanceInfo(
   printJob : Smallint;
   alertN : Smallint;
   instanceN : DWord;
   var alertInstanceInfo : PEAlertInstanceInfo) : boolean stdcall;
```

PEGetNthFormula

Use PEGetNthFormula to retrieve information about a specific formula in the report. Use this function to obtain the formula name and formula text of a specific formula in the report. This function can be used to retrieve the formula text to allow the user to edit the formula. You can then change the formula text with "PESetFormula" on page 405.

C Syntax

```
BOOL CRPE_API PEGetNthFormula (
        short printJob,
        short formulaN,
        HANDLE FAR *nameHandle,
        short FAR *nameLength,
        HANDLE FAR *textHandle,
        short FAR *textLength );
```

Parameters

printJob	Specifies the print job from which you want to gather formula information.
formulaN	Specifies the 0-based number of the formula about which you want to gather information.
nameHandle	Specifies a pointer to the handle of the string containing the formula name.
nameLength	Specifies a pointer to the length of the formula name string (in bytes) including the terminating byte.
textHandle	Specifies a pointer to the handle of the string containing the formula text.
textLength	Specifies a pointer to the length of the formula string (in bytes) including the terminating byte.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Edit | Formula

VB Syntax

```
Declare Function PEGetNthFormula Lib "crpe32.dll" (ByVal printJob As
Integer, ByVal FormulaN As Integer, NameHandle As Long, NameLength As
Integer, TextHandle As Long, TextLength As Integer) As Integer
```

Delphi Syntax

```
function PEGetNthFormula (
    printJob: Word;
    formulaN: integer;
    var nameHandle: Hwnd;
    var nameLength: Word;
    var textHandle: Hwnd;
    var textLength: Word
    ): Bool stdcall;
```

PEGetNthGroupSortField

Use PEGetNthGroupSortField to retrieve information about one of the group sort fields in the specified report. This function is used with "PEGetHandleString" on page 318. PEGetNthGroupSortField returns the name of the field and the direction (ascending or descending) of the sort. See "Crystal Report Engine API variable length strings" on page 51 for additional information.

This function is typically used as one of a series of functions ("PEGetNGroupSortFields" on page 323) called once; PEGetNthGroupSortField *and* "PEGetHandleString" on page 318 called as many times as needed to identify the correct group sort field; and "PESetNthAlertConditionFormula" on page 418 called once when the correct sort field is identified. The series can be used in a Custom-Print Link to identify and then change an existing group sort field and / or sort order in response to a user selection at print time.

C Syntax

```
BOOL CRPE_API PEGetNthGroupSortField (
        short printJob,
        short sortFieldN,
        HANDLE FAR *nameHandle,
        short FAR *nameLength,
        short FAR *direction );
```

Parameters

printJob	Specifies the print job from which you want to gather group sort field information.
sortFieldN	Specifies the 0-based number of the group sort field that you want to retrieve. The first group sort field is field 0. If the report has N sort fields, the function can be called with sortFieldN between 0 and N-1.
nameHandle	Specifies a pointer to the handle of the string containing the sort field name.
nameLength	Specifies a pointer to the length of the field name string (in bytes) including the terminating byte.
direction	Specifies a pointer to the sort direction. Uses one of the PE_SF_XXX "Sort Order Constants" on page 560.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

To find out top / bottom n group sort information, use "PEGetGroupOptions" on page 316.

VB Syntax

```
Declare Function PEGetNthGroupSortField Lib "crpe32.dll" ( _
    ByVal printJob As Integer, ByVal SortFieldN As Integer, _
    NameHandle As Long, NameLength As Integer, Direction As Integer _
    ) As Integer
```

Delphi Syntax

```
function PEGetNthGroupSortField (
    printJob: Word;
     sortFieldN: integer;
     var nameHandle: HWnd;
     var nameLength: Word;
     var direction: Word
    ): Bool stdcall;
```

PEGetNthParameterCurrentRange

Use PEGetNthParameterCurrentRange to retrieve a value range from the specified parameter field in a report. Use "PEGetNParameterCurrentRanges" on page 325, to get the number of value ranges currently associated with the parameter field. See "Working with Parameter Values and Ranges" on page 45.

C Syntax

```
BOOL CRPE_API PEGetNthParameterCurrentRange (
        short printJob,
        const char FAR *parameterFieldName,
        const char FAR *reportName,
        short index,
        PEValueInfo FAR *rangeStart,
        PEValueInfo FAR *rangeEnd,
        short FAR *rangeInfo );
```

Parameters

printJob	Specifies the print job for which you want to retrieve the parameter current range.
parameterFieldName	Specifies a pointer to the string containing the parameter field name.
reportName	Specifies a pointer to the string containing the report name. See Remarks below.
index	Specifies the index number of the value range to be retrieved.
rangeStart	Specifies a pointer to "PEValueInfo" on page 516, in which the beginning value in the range is returned.
rangeEnd	Specifies a pointer to "PEValueInfo" on page 516, in which the final value in the range is returned.
rangeInfo	Use this bitwise value to indicate whether the upper and/or lower bound(s) in the range should be retrieved. Use one or more of the "Range Info Constants" on page 559.

Returns

- TRUE if the call succeeds.
- FALSE if the call fails.

Remarks

Regarding parameter reportName:

- For the main report, pass an empty string ("").
- For a subreport, pass the file path and name of the subreport as a NULL-terminated string.

VB Syntax

```
Declare Function PEGetNthParameterCurrentRange Lib "crpe32.dll" ( _
    ByVal printJob As Integer, ByVal parameterFieldName As String, _
    ByVal reportName As String, ByVal index As Integer, _
    rangeStart As PEValueInfo, rangeEnd As PEValueInfo, _
    ByVal rangeInfo As Integer ) As Integer
```

Delphi Syntax

```
procedure PEGetNthParameterCurrentRange (
    printJob: smallint;
    const parameterFieldName: PChar;
    const reportName: PChar;
    index: smallint;
    var rangeStart: PEValueInfo;
    var rangeEnd: PEValueInfo;
    rangeInfo: smallint
    ): BOOL stdcall;
```

PEGetNthParameterCurrentValue

Use to retrieve a value from the specified parameter field of a report. Use "PEGetNParameterCurrentValues" on page 325, to determine the number of values currently held in the parameter field. See "Working with Parameter Values and Ranges" on page 45.

C Syntax

```
BOOL CRPE_API PEGetNthParameterCurrentValue (
        short printJob,
        const char FAR *parameter FieldName,
        const char FAR *reportName,
        short index,
        PEValueInfo FAR *currentValue );
```

Parameters

printJob	Specifies the print job for which you want to determine the parameter current value.
parameterFieldName	Specifies a pointer to a string containing the parameter field name.
reportName	Specifies a pointer to a string containing the report name. See Remarks below.
index	Specifies the index number of the value to be retrieved.
currentValue	Specifies a pointer to "PEValueInfo" on page 516, in which the value will be returned. If it contains no value then it will be set to the constant PE_VI_NOVALUE rather than NULL.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

Regarding parameter reportName:

- For the main report, pass an empty string ("").
- For a subreport, pass the file path and name of the subreport as a NULL-terminated string.

VB Syntax

```
Declare Function PEGetNthParameterCurrentValue Lib "crpe32.dll" ( _
    ByVal printJob As Integer, ByVal parameterFieldName As String, _
    ByVal reportName As String, ByVal index As Integer, _
    currentValue As PEValueInfo ) As Integer
```

Delphi Syntax

```
function PEGetNthParameterCurrentValue (
    printJob: smallint;
    const parameterFieldName: PChar;
    const reportName: PChar;
    index: smallint;
    var currentValue: PEValueInfo
    ): BOOL stdcall;
```

PEGetNthParameterDefaultValue

Use PEGetNthParameterDefaultValue to retrieve a default value for a specified parameter field in a report. Use "PEGetNParameterDefaultValues" on page 326, to retrieve the number of default values for the parameter field.

C Syntax

```
BOOL CRPE_API PEGetNthParameterDefaultValue (
        short printJob,
        const char FAR *parameterFieldName,
        const char FAR *reportName,
        short index,
        PEValueInfo FAR *valueInfo );
```

Parameters

printJob	Specifies the print job from which you want to gather parameter default value information.
parameterFieldName	Specifies a pointer to the string containing the parameter field name.
reportName	Specifies a pointer to the string containing the report name. See Remarks below.
index	Specifies the index number of the default value to be retrieved.
valueInfo	Specifies a pointer to “PEValueInfo” on page 516, containing information about requested default value.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

Regarding parameter reportName:

- For the main report, pass an empty string ("").
- For a subreport, pass the file path and name of the subreport as a NULL-terminated string.

VB Syntax

```
Declare Function PEGetNthParameterDefaultValue Lib "crpe32.dll" ( _
    ByVal printJob As Integer, ByVal parameterFieldName As String, _
    ByVal reportName As String, ByVal index As Integer, _
    valueInfo As PEValueInfo ) As Integer
```

Delphi Syntax

```
function PEGetNthParameterDefaultValue(
    printJob: smallint;
    const parameterFieldName: PChar;
    index: smallint;
    var valueInfo: PEValueInfo
    ): BOOL stdcall;
```

PEGetNthParameterField

Use PEGetNthParameterField to retrieve information about one of the parameter fields in the specified report. For new development, see Remarks below. This function returns the name of the field, the data type, and information about the value set for the field. The name of the parameter field is returned as a string handle. This function is typically used as one of a series of functions ("PEGetNParameterFields" on page 327, (called once); PEGetNthParameterField (called as many times as needed to identify the correct parameter field); and "PESetNthParameterField" on page 423, (called once when the correct parameter field is identified). The series can be used in a Custom-Print Link to identify and then change an existing parameter field value in response to a user selection at print time.

C Syntax

```
BOOL CRPE_API PEGetNthParameterField (
        short printJob,
        short parameterN,
        PEParameterFieldInfo FAR *parameterInfo );
```

Parameters

printJob	Specifies the print job that contains the parameter field about which you want to retrieve information.
parameterN	Specifies the number of the parameter field about which you want to retrieve information.
parameterInfo	Specifies a pointer to "PEParameterFieldInfo" on page 484, which is used to store the information you retrieve. See Remarks below.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

- For new development, the Default/CurrentValueSet members of PEParameterFieldInfo must be set to FALSE and the following new calls used to access default and current value lists.
  - "PEGetNParameterCurrentRanges" on page 325
  - "PEGetNParameterCurrentValues" on page 325
  - "PEGetNParameterDefaultValues" on page 326
  - "PEGetNthParameterCurrentRange" on page 336
  - "PEGetNthParameterCurrentValue" on page 337
  - "PEGetNthParameterDefaultValue" on page 338
- The CurrentValue member of the returned structure, "PEParameterFieldInfo" on page 484, will be set to CRWNULL. if the parameter is NULL.

VB Syntax

```
Declare Function PEGetNthParameterField Lib "crpe32.dll" (ByVal printJob
As Integer, ByVal varN As Integer, varInfo As PEParameterFieldInfo) As
Integer
```

Delphi Syntax

```
function PEGetNthParameterField (
    printJob: Word;
    varN: Smallint;
    var varInfo: PEParameterFieldInfo
    ): Bool stdcall;
```

PEGetNthParameterType

Use PEGetNthParameterType to retrieve the type (or origin) of a specified parameter.

C Syntax

```
short CRPE_API PEGetNthParameterType (
        short printJob,
        short index );
```

Parameters

printJob	Specifies the print job from which you want to gather parameter type information.
index	Specifies the index number of the parameter field.

Returns

Returns one of the following PE_PO_XXX Constants or -1 if the index is invalid.

PE_PO_REPORT	Report.
PE_PO_STOREDPROC	Stored Procedure.
PE_PO_QUERY	Query.

VB Syntax

```
Declare Function PEGetNthParameterType Lib "crpe32.dll" ( _
    ByVal printJob As Integer, ByVal index As Integer ) As Integer
```

Delphi Syntax

```
function PEGetNthParameterType (
    printJob: smallint;
    index: smallint
    ): smallint stdcall;
```

PEGetNthParameterValueDescription

Use PEGetNthParameterValueDescription to retrieve the description of the value set for a parameter.

C Syntax

```
BOOL CRPE_API PEGetNthParameterValueDescription (
        short printJob,
        const char FAR *parameterFieldName,
        const char FAR *reportName,
        short index,
        HANDLE FAR *valueDesc,
        short FAR *valueDescLength );
```

Parameters

printJob	Specifies the print job from which you want to gather parameter value description information.
parameterFieldName	Specifies a pointer to the parameterFieldName for which you want to retrieve the parameter value description.
reportName	Specifies a pointer to the report name. See Remarks below.
index	Specifies the index.
valueDesc	Specifies a pointer to the handle of the value description to be retrieved.
valueDescLength	Specifies a pointer to the length of the value description.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

Regarding parameter reportName:

- For the main report, pass an empty string ("").
- For a subreport, pass the file path and name of the subreport as a NULL-terminated string.

VB Syntax

```
Declare Function PEGetNthParameterValueDescription Lib "crpe32.dll" (_
    ByVal printJob As Integer, ByVal parameterFieldName As String, _
    ByVal reportName As String, ByVal index As Integer, _
    valueDesc As Long, valueDescLength As Integer ) As Integer
```

Delphi Syntax

```
function PEGetNthParameterValueDescription (
  printJob            : Smallint;
  parameterFieldName  : PChar;
  reportName          : PChar;
  index               : Smallint;
  var valueDesc       : HWnd;
  var valueDescLength : Smallint): Bool; {$ifdef WIN32} stdcall; {$endif}
```

PEGetNthReportAlert

Use PEGetNthReportAlert to retrieve the PEReportAlertInfo structure from the report associated with the job handle. The PEReportAlertInfo structure contains information on a specified Report Alert.

C Syntax

```
BOOL CRPE_API PEGetNthReportAlert (short printJob,
                                   short alertN,
                                   PEReportAlertInfo FAR *
reportAlertInfo);
```

Parameters

printJob	Specifies the print job from which you want to get the number of Report Alerts.
alertN	Specifies the Report Alert from which you want to get the alert instance information.
reportalertInfo	Specifies a pointer to “PEAlertInstanceInfo” on page 453

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEGetNthReportAlert Lib "crpe32.dll" (ByVal printJob%,
ByVal alertN%, reportAlertInfo As PEReportAlertInfo) As Integer
```

Delphi Syntax

```
function PEGetNthReportAlert(
   printJob : Smallint;
   alertN : Smallint;
   var reportAlertInfo : PEReportAlertInfo) : boolean stdcall;
```

PEGetNthSortField

Use PEGetNthSortField to return information about one of the sort fields in the specified report. This function returns the name of the field and the direction (ascending or descending) of the sort. The name of the sort field is returned as a string handle. This function is typically used as one of a series of functions ("PEGetNSortFields" on page 330 called once; PEGetNthSortField and "PEGetHandleString" on page 318 called together as many times as needed to identify the correct sort field; and "PESetNthSortField" on page 425 called once when the correct sort field is identified). The series can be used in a Custom-Print Link to identify and then change an existing sort field and / or sort order in response to a user selection at print time.

C Syntax

```
BOOL CRPE_API PEGetNthSortField (
        short printJob,
        short sortFieldN,
        HANDLE FAR *nameHandle,
        short FAR *nameLength,
        short FAR *direction );
```

Parameters

printJob	Specifies the print job from which you want to retrieve sort field information.
sortFieldN	Specifies the0-based number of the sort field you want to retrieve. The first sort field is field 0. If the report has N sort fields, the function can be called with sortFieldN between 0 and N-1.
nameHandle	Specifies a pointer to the handle of the string containing the sort field name.
nameLength	Specifies a pointer to the length of the field name string (in bytes) including the terminating byte.
direction	Specifies a pointer to the sort direction. Uses one of the PE_SF_XXX "Sort Order Constants" on page 560.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEGetNthSortField Lib "crpe32.dll" ( _
    ByVal printJob As Integer, ByVal SortNumber As Integer, _
    NameHandle As Long, NameLength As Integer, Direction As Integer _
    ) As Integer
```

Delphi Syntax

```
function PEGetNthSortField (
    printJob: Word;
    sortFieldN: integer;
    var nameHandle: HWnd;
    var nameLength: Word;
    var direction: Word
    ): Bool stdcall;
```

PEGetNthSQLExpression

Use PEGetNthSQLExpression to retrieve one of the SQL expressions associated with a report. Use this function with "PEGetHandleString" on page 318. Use "PEGetNSQLExpressions" on page 330, to determine the number of SQL expressions in the report.

C Syntax

```
BOOL CRPE_API PEGetNthSQLExpression (
        short printJob,
        short expressionN,
        HANDLE FAR *nameHandle,
        short FAR *nameLength,
        HANDLE FAR *textHandle,
        short FAR *textLength );
```

Parameters

printJob	Specifies the print job from which you want to gather SQL expression information.
expressionN	Specifies the numeric value indicating which expression to retrieve.
nameHandle	Specifies a pointer to the handle of the string containing the expression name.
nameLength	Specifies a pointer to the length of name string.
textHandle	Specifies a pointer to the handle of the string containing the SQL expression.
textLength	Specifies a pointer to the length of the expression string.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEGetNthSQLExpression Lib "crpe32.dll" (ByVal printJob
As Integer, ByVal expressionN As Integer, nameHandle As Long, nameLength
As Integer, textHandle As Long, textLength As Integer) As Integer
```

Delphi Syntax

```
function PEGetNthSQLExpression (
    printJob: smallint;
    expressionN: Smallint;
    var nameHandle: Hwnd;
    var nameLength: Smallint;
    var textHandle: Hwnd;
    var textLength: Smallint
    ): Bool stdcall;
```

PEGetNthSubreportInSection

Use PEGetNthSubreportInSection to retrieve a handle that is required to retrieve the name of a subreport.

Syntax

```
DWORD CRPE_API PEGetNthSubreportInSection (
        short printJob,
        short sectionCode,
        short subreportN );
```

Parameters

printJob	Specifies the primary report.
sectionCode	Specifies the “Section Codes” on page 559, for the report section that contains the subreport. See “Working with section codes” on page 46.
subreportN	Specifies the number of the subreport in the specified section. subreportN is zero based. The first report in the section will be 0, the second will be 1, etc. If there are no subreports in the section, the function will return 0.

Returns

Returns a handle that is used to retrieve the name of the specified subreport.

Remarks

Use “PEGetSubreportInfo” on page 367, to retrieve information about the subreport by passing the subreport handle returned by PEGetNthSubreportInSection.

VB Syntax

```
Declare Function PEGetNthSubreportInSection Lib “crpe32.dll” (
    ByVal printJob As Integer, ByVal sectionCode As Integer,
    ByVal subreportN As Integer ) As Long
```

Delphi Syntax

```
function PEGetNthSubreportInSection (
    printJob: Word;
```

```
    sectionCode: Smallint;
    subreportN: Smallint
    ): DWORD stdcall;
```

PEGetNthTableLocation

Use PEGetNthTableLocation to determine the location of a selected table used in the specified print job. This function is typically combined with "PESetNthTableLocation" on page 426 to identify the location of a table and then to change it.

C Syntax

```
BOOL CRPE_API PEGetNthTableLocation (
        short printJob,
        short tableN,
        PETableLocation FAR *location );
```

Parameters

printJob	Specifies the print job from which you want to retrieve information about a table's location.
tableN	Specifies the 0-based number of the table for which you want to retrieve table location information. The first table is table 0. The last table is N-1.
location	Specifies the pointer to "PETableLocation" on page 510. The format of the string will be depend on the type of database specified.

Returns

- TRUE if the call is successful0.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEGetNthTableLocation Lib "crpe32.dll" ( ByVal printJob
As Integer, ByVal TableN As Integer, Location As PETableLocation ) As
Integer
```

Delphi Syntax

```
function PEGetNthTableLocation(
    printJob: Word;
    tableN: integer;
    var location: PETableLocation
    ): Bool stdcall;
```

PEGetNthTableLogOnInfo

Use PEGetNthTableLogOnInfo to retrieve log on information required by a report.

C Syntax

```
BOOL CRPE_API PEGetNthTableLogOnInfo (
        short printJob,
        short tableN,
        PELogOnInfo FAR *logOnInfo );
```

Parameters

printJob	Specifies the print job for which you want to get table log on information.
tableN	Specifies the 0-based number of the table for which you want to retrieve table log on information. The first table is table 0. The last table is N-1.
logOnInfo	Specifies the pointer to "PELogOnInfo" on page 479.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

- This function must be called after "PEOpenPrintJob" on page 378 (so you have the job handle), and before "PEStartPrintJob" on page 448 (which needs the password set to print the report).
- The password in "PELogOnInfo" on page 479 will always be an empty string when using this function.

VB Syntax

```
Declare Function PEGetNthTableLogOnInfo Lib "crpe32.dll" ( ByVal
printJob As Integer, ByVal TableN As Integer, LogOnInfo As PELogOnInfo )
As Integer
```

Delphi Syntax

```
function PEGetNthTableLogOnInfo (
    printJob: Word;
    tableN: integer;
    var logOnInfo: PELogOnInfo
    ): Bool stdcall;
```

PEGetNthTablePrivateInfo

Retrieves information for using data objects such as ADO, RDO, or CDO with the Active Data Driver (PS2MON.DLL).

C Syntax

```
BOOL CRPE_API PEGetNthTablePrivateInfo (
        short printJob,
        short tableN,
        PETablePrivateInfo FAR *privateInfo );
```

Parameters

printJob	Identifies the print job which uses the table for which you want to retrieve the session information.
tableN	Specifies the 0-based number of the table for which you want to retrieve table private information. The first table is table 0. The last table is N-1.
privateInfo	Specifies the pointer to “PETablePrivateInfo” on page 511.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Delphi Syntax

```
function PESetNthTablePrivateInfo (
    printJob: Smallint;
    tableN:   Smallint;
    var privateInfo:          PETablePrivateInfo
)   :Bool; {$ifdef WIN32} stdcall; {$endif}
```

PEGetNthTableSessionInfo

Use PEGetNthTableSessionInfo to set the session information for a Microsoft Access table being used in your report. Many MS Access database tables require that a session be opened before the information in the table can be used. Use PEGetNthTableSessionInfo to obtain the session information (User ID, Password, and Session Handle) for a particular table.

C Syntax

```
BOOL CRPE_API PEGetNthTableSessionInfo (
        short printJob,
        short tableN,
        PESessionInfo FAR *sessionInfo );
```

Parameters

printJob	Identifies the print job that uses the table for which you want to retrieve the session information.
tableN	Specifies the 0-based number of the table for which you want to retrieve session information. The first table is table 0. The last table is N-1.
sessionInfo	Specifies a pointer to "PESessionInfo" on page 503.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

- This function is only applicable for MS Access databases which require a session to be opened before the database is accessed.
- The password in "PESessionInfo" on page 503 will always be an empty string when using this function.

VB Syntax

```
Declare Function PEGetNthTableSessionInfo Lib "crpe32.dll" ( _
    ByVal printJob As Integer, ByVal TableN As Integer, _
    SessionInfo As PESessionInfo ) As Integer
```

Delphi Syntax

```
function PEGetNthTableSessionInfo (
    printJob: Word;
    tableN: Integer;
    var sessionInfo: PESessionInfo
    ): Bool stdcall;
```

PEGetNthTableType

Use PEGetNthTableType to determine the type of each table. This function is one of a series of functions that enable you to retrieve and update database information in an opened report so that the report can be printed using different server, database, user, and/or table location settings.

Syntax

```
BOOL CRPE_API PEGetNthTableType (
        short printJob,
        short tableN,
        PETableType FAR *tableType );
```

Parameters

printJob	Specifies the print job for which you want to determine a table type.
tableN	Specifies the 0-based number of the table for which you want to determine the type. Table numbers start at 0. For example, if PEGetNTables returns 2, call PEGetNthTableType twice with table numbers of 0 and 1.
tableType	Specifies a pointer to “PETableType” on page 512.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

- The application can test DBType in “PETableType” on page 512 or test the database DLL name used to create the report. DBType is the structure returned by PEGetNthTableType.
  - DLL names have the following naming convention:

  PDB*.DLL for standard (non-SQL) databases.

  PDS*.DLL for SQL/ODBC databases.
  - In the case of ODBC (PDSODBC.DLL) the DescriptiveName includes the ODBC data source name.
- PEGetNthTableType must be called after PEOpenPrintJob and before PEStartPrintJob.

VB Syntax

```
Declare Function PEGetNthTableType Lib “crpe32.dll” ( _
    ByVal printJob As Integer, ByVal TableN As Integer, _
    TableType As PETableType ) As Integer
```

Delphi Syntax

```
function PEGetNthTableType (
    printJob: Word;
    tableN: Integer;
    var tableType: PETableType
    ): Bool stdcall;
```

PEGetParameterMinMaxValue

Use PEGetParameterMinMaxValue to retrieve the minimum and/or maximum possible values for a specified parameter in a report.

C Syntax

```
BOOL CRPE_API PEGetParameterMinMaxValue (
        short printJob,
```

```
    const char FAR *parameterFieldName,
    const char FAR *reportName,
    PEValueInfo FAR *valueMin,
    PEValueInfo FAR *valueMax );
```

Parameters

printJob	Specifies the print job for which you want to retrieve the parameter min/max values.
parameterFieldName	Specifies a pointer to the string containing the parameter name.
reportName	Specifies a pointer to the string containing the report name. See Remarks below.
valueMin	Specifies a pointer to "PEValueInfo" on page 516, in which information about the minimum parameter value will be returned. See Remarks below.
valueMax	Specifies a pointer to "PEValueInfo" on page 516, in which information about the maximum parameter value will be returned. See Remarks below.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

- Regarding parameter reportName:
  - For the main report, pass an empty string ("").
  - For a subreport, pass the file path and name of the subreport as a NULL-terminated string.
- Regarding parameters valueMin and valueMax:
  - Set valueMin to NULL to retrieve maximum value only; must be non-NULL if valueMax is NULL.
  - Set valueMax to NULL to retrieve minimum value only; must be non-NULL if valueMin is NULL.

VB Syntax

```
Declare Function PEGetParameterMinMaxValue Lib "crpe32.dll" ( _
    ByVal printJob As Integer, ByVal parameterFieldName As String, _
    ByVal reportName As String, valueMin As PEValueInfo, _
    valueMax As PEValueInfo ) As Integer
```

Delphi Syntax

```
function PEGetParameterMinMaxValue (
    printJob: smallint;
    const parameterFieldName: PChar;
    const reportName: PChar;
```

```
    var valueMin: PEValueInfo;
    var valueMax: PEValueInfo }
    ): BOOL stdcall;
```

PEGetParameterPickListOption

Use PEGetParameterPickListOption to retrieve the parameter pick list options set for a report. This function retrieves the values in "PEParameterPickListOption" on page 487.

C Syntax

```
BOOL CRPE_API PEGetParameterPickListOption (
        short printJob,
        const char FAR *parameterFieldName,
        const char FAR *reportName,
        PEParameterPickListOption FAR *pickListOption );
```

Parameters

printJob	Specifies the print job from which you want to retrieve the parameter pick list options.
parameterFieldName	Specifies a pointer to the parameterFieldName for which you want to retrieve pick list options.
reportName	Specifies a pointer to the report name. See Remarks below.
pickListOption	Specifies a pointer to "PEParameterPickListOption" on page 487, which will contain the information retrieved.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

Regarding parameter reportName:

- For the main report, pass an empty string ("").
- For a subreport, pass the file path and name of the subreport as a NULL-terminated stringand name of the subreport as a NULL-terminated string.

VB Syntax

```
Declare Function PEGetParameterPickListOption Lib "crpe32.dll" (_
    ByVal printJob As Integer, ByVal parameterFieldName As String, _
    ByVal reportName As String, pickListOption As
PEParameterPickListOption _
    ) As Integer
```

Delphi Syntax

```
function PEGetParameterPickListOption (
  printJob           : Smallint;
  parameterFieldName : PChar;
  reportName         : PChar;
  var pickListOption : PEParameterPickListOption): Bool; {$ifdef WIN32}
        stdcall; {$endif}
```

PEGetParameterValueInfo

Use PEGetParameterValueInfo to retrieve the "PEParameterValueInfo" on page 488, structure associated with the specified parameter field in a report. This structure contains information (for example, editing possible, nullable field, multiple values, etc.) about the values which can be stored in this field. See "Working with Parameter Values and Ranges" on page 45.

C Syntax

```
BOOL CRPE_API PEGetParameterValueInfo (
        short printJob,
        const char FAR *paramterFieldName,
        const char FAR *reportName,
        PEParameterValueInfo FAR *valueInfo );
```

Parameters

printJob	Specifies the print job from which you want to gather parameter value information.
parameterField Name	Specifies a pointer to the string containing the parameter field name.
reportName	Specifies a pointer to the string containing the report name. See Remarks below.
valueInfo	Specifies a pointer to "PEParameterValueInfo" on page 488, in which the parameter value information will be returned. See "Working with Parameter Values and Ranges" on page 45.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

Regarding parameter reportName:

- For the main report, pass an empty string ("").
- For a subreport, pass the file path and name of the subreport as a NULL-terminated string.

VB Syntax

```
Declare Function PEGetParameterValueInfo Lib "crpe32.dll" ( _
    ByVal printJob As Integer, ByVal parameterFieldName As String, _
    ByVal reportName As String, valueInfo As PEParameterValueInfo ) As
Integer
```

Delphi Syntax

```
function PEGetParameterValueInfo (
    printJob : smallint;
    const parameterFieldName : PChar;
    const reportNam : PChar;
    var valueInfo : PEParameterValueInfo
    ): BOOL stdcall;
```

PEGetPrintDate

Use PEGetPrintDate to determine the print date (if any) that was specified with the report. Use this function to retrieve the print date and pass it back using "PESetPrintDate" on page 434.

C Syntax

```
BOOL CRPE_API PEGetPrintDate (
        short printJob,
        short FAR *year,
        short FAR *month,
        short FAR *day );
```

Parameters

printJob	Specifies the print job for which you want to retrieve the print date setting.
year	Specifies a pointer to the year component of the print date.
month	Specifies a pointer to the month component of the print date.
day	Specifies a pointer to the day component of the print date.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

You change the print date, typically, when you want to run the report today yet have it appear to have been run on a different date. An example would be, if you were out of town on the last day of the previous month and you later want to run a report for that month and make it appear as if it were run on the last day of the month rather than the current date.

VB Syntax

```
Declare Function PEGetPrintDate Lib "crpe32.dll" (ByVal printJob As
Integer, Date_Year As Integer, Date_Month As Integer, Date_Day As
Integer) As Integer
```

Delphi Syntax

```
function PEGetPrintDate (
    printJob: Word;
    var year: Word;
    var month: Word;
    var day: Word
    ): Bool stdcall;
```

PEGetPrintOptions

Use PEGetPrintOptions to retrieve the print options specified for the report (the options that are set in the Print common dialog box) and use them to fill in "PEPrintOptions" on page 489. Use this function to retrieve print options from the report in order to update them and pass back using "PESetPrintOptions" on page 435.

C Syntax

```
BOOL CRPE_API PEGetPrintOptions (
        short printJob,
        PEPrintOptions FAR *options );
```

Parameters

printJob	Specifies the print job that you want to query to determine which print options have been set using the Print common dialog box.
options	Specifies a pointer to "PEPrintOptions" on page 489.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEGetPrintOptions Lib “crpe32.dll” (
    ByVal printJob As Integer, Options As PEPrintOptions ) As Integer
```

Delphi Syntax

```
function PEGetPrintOptions (
    printJob: Word;
    var options: PEPrintOptions
    ): Bool stdcall;
```

PEGetReportOptions

Use PEGetReportOptions to retrieve the report options associated with a specified print job.

C Syntax

```
BOOL CRPE_API PEGetReportOptions (
        short printJob,
        PEReportOptions FAR *reportOptions );
```

Parameters

printJob	Specifies the print job for which you want to determine report options.
reportOptions	Specifies a pointer to “PEReportAlertInfo” on page 494, which contains report options for the specified print job.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEGetReportOptions Lib "crpe32.dll" (ByVal printJob _
        As Integer, reportOptions As PEReportOptions) As Integer
```

Delphi Syntax

```
function PEGetReportOptions (
    printJob: smallint;
    var reportOptions: PEReportOptions
    ): Bool stdcall;
```

PEGetReportSummaryInfo

Use PEGetReportSummaryInfo to retrieve summary information about the report (for example, report title, author, and comments).

C Syntax

```
BOOL CRPE_API PEGetReportSummaryInfo (
 short printJob,
 PEReportSummaryInfo FAR *summaryInfo);
```

Parameters

printJob	Specifies the print job for which you want to get the report summary information.
summaryInfo	Specifies a pointer to "PEReportSummaryInfo" on page 499.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEGetReportSummaryInfo Lib "crpe32.dll" (
    ByVal printJob As Integer, summaryInfo As PEReportSummaryInfo ) As
Integer
```

Delphi Syntax

```
function PEGetReportSummaryInfo (
    printJob: integer;
    var summaryInfo: PEReportSummaryInfo
    ): Bool stdcall;
```

PEGetReportTitle

Use PEGetReportTitle to retrieve the handle of the report title string in the report summary information. If the job is a subreport, it returns the handle of the subreport name. Use this function with "PEGetHandleString" on page 318. Use "PESetReportTitle" on page 437, to pass back a report title. The series can be used in Custom-Print Links to identify and change an existing report title in response to a user selection at print time.

C Syntax

```
BOOL CRPE_API PEGetReportTitle (
        short printJob,
        HANDLE FAR *titleHandle,
        short FAR *titleLength );
```

Parameters

printJob	Specifies the print job for which you want to get the report title.
titleHandle	Specifies a pointer to the handle of the title string.
titleLength	Specifies a pointer to the length of the title string (in bytes) including the terminating byte.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.l

VB Syntax

```
Declare Function PEGetReportTitle Lib "crpe32.dll" (
    ByVal printJob As Integer, TitleHandle As Long, TitleLength As
Integer
    ) As Integer
```

Delphi Syntax

```
function PEGetReportTitle (
    printJob: Word;
    var titleHandle: HWnd;
    var titleLength: Word
    ): Bool stdcall;
```

PEGetReportVersion

Use PEGetReportVersion to retrieve the PEVersionInfo structure associated with the print job. The PEVersionInfo structure contains the reports version information.

C Syntax

```
BOOL CRPE_API PEGetReportVersion( short printJob,
                                  PEVersionInfo FAR* pVersionInfo )
```

Parameters

printJob	Specifies the print job from which you want to get the report version info.
pVersionInfo	Specifies a pointer to "PEVersionInfo" on page 518

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEGetReportVersion Lib "crpe32.dll" (ByVal printJob%,
pVersionInfo As PEVersionInfo) As Integer
```

Delphi Syntax

```
function PEGetReportVersion(
   printJob : Smallint;
   var pVersionInfo : PEVersionInfo): Bool stdcall;
```

PEGetSectionCode

Use PEGetSectionCode to retrieve the section code for the specified section. A section code indicates the section type (Page Header, Details, etc.). If there are multiple group sections it also identifies the group number, and if there are multiple sections in an area it identifies the section number. See "Working with section codes" on page 46.

C Syntax

```
short CRPE_API PEGetSectionCode (
        short printJob,
        short sectionN );
```

Parameters

printJob	Specifies the print job from which you want to retrieve a section code.
sectionN	Specifies the number of the section for which you want the section code. This parameter should be within the range obtained by "PEGetNSections" on page 328.

Returns

- Returns the "Section Codes" on page 559, for the specified section.
- Returns 0 if the call fails.

VB Syntax

```
Declare Function PEGetSectionCode Lib "crpe32.dll" (
    ByVal printJob As Integer, ByVal sectionN As Integer ) As Integer
```

Delphi Syntax

```
function PEGetSectionCode (
    printJob: Word;
    sectionN: Smallint
    ): Smallint stdcall;
```

PEGetSectionFormat

Use PEGetSectionFormat to retrieve the section format settings for selected sections in the specified report and supply them as member values for “PESectionOptions” on page 501. Use this function in order to update the section formats and pass them back using “PESetSectionFormat” on page 438.

C Syntax

```
BOOL CRPE_API PEGetSectionFormat (
        short printJob,
        short sectionCode,
        PESectionOptions FAR *options );
```

Parameters

printJob	Specifies the print job that you want to query to determine what report section options have been set using the Format Section dialog box.
sectionCode	Specifies the“Section Codes” on page 559, for the report section(s) for which you want to get section format information. See “Working with section codes” on page 46.
options	Specifies a pointer to “PESectionOptions” on page 501.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEGetSectionFormat Lib “crpe32.dll” (
    ByVal printJob As Integer, ByVal sectionCode As Integer,
    Options As PESectionOptions ) As Integer
```

Delphi Syntax

```
function PEGetSectionFormat (
    printJob: Word;
    sectionCode: integer;
    var options: PESectionOptions
    ): Bool stdcall;
```

PEGetSectionFormatFormula

Use PEGetSectionFormatFormula to retrieve the current format formula for the specified section of the report. Use this function with “PEGetHandleString” on page 318.

C Syntax

```
BOOL CRPE_API PEGetSectionFormatFormula (
        short printJob,
        short sectionCode,
        short formulaName,
        HANDLE FAR *textHandle,
        short FAR *textLength );
```

Parameters

printJob	Specifies the print job for which you want to retrieve the section format formula.
sectionCode	Specifies the "Section Codes" on page 559, for the report section(s) for which you want to get section format information. See "Working with section codes" on page 46.
formulaName	Specifies the name of the section format formula. Use one of the PE_FFN_XXX "Area/Section Format Formula Constants" on page 541.
textHandle	Specifies a pointer to the handle of the text of the actual formula.
textLength	Specifies a pointer to the length of the text string. Use this value to allocate a buffer for the text.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

- Not all formula names apply to all sections.
- Use the value returned by textLength to allocate memory for a buffer. Use "PEGetHandleString" on page 318, to fill the buffer with the actual text of the formula.

VB Syntax

```
Declare Function PEGetSectionFormatFormula Lib "crpe32.dll" (
    ByVal printJob As Integer, ByVal sectionCode As Integer,
    ByVal formulaName As Integer, textHandle As Long,
    textLength As Integer ) As Integer
```

Delphi Syntax

```
function PEGetSectionFormatFormula (
    printJob: Word;
    sectionCode: Word;
    formulaName: Word;
    var textHandle: HWnd;
    var textLength: Word
    ): Bool stdcall;
```

PEGetSectionHeight

Use PEGetSectionHeight to retrieve the section height information for the specified section. This is the replacement API Call for *PEGetMinimumSectionHeight* and should be used for all new development.

C Syntax

```
BOOL CRPE_API PEGetSectionHeight (
        short printJob,
        short sectionCode,
        short FAR *height );
```

Parameters

printJob	Specifies the print job that you want to query to retrieve section height information.
section Code	Specifies the "Section Codes" on page 559, for the report sections for which you want to retrieve information. See "Working with section codes" on page 46.
height	Specifies a pointer to the section height (in twips).

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEGetSectionHeight Lib "crpe32.dll" (
    ByVal printJob As Integer, ByVal sectionCode As Integer,
    height As Integer ) As Integer
```

Delphi Syntax

```
function PEGetSectionHeight (
    printJob: Smallint;
    sectionCode: Smallint;
    Height: Smallint (*in twips*)
    ): Bool stdcall;
```

PEGetSelectedPrinter

Use PEGetSelectedPrinter to obtain information about the printer currently selected for the report. If a printer has been specified in Crystal Reports using the File | Printer Setup | Specific printer option, this call will return information about that printer. If the File | Printer Setup | Default printer option has been selected for the report, and custom options for the Default printer have been specified (Default Properties is toggled off in the Print Setup dialog box), information about the default printer specified under Windows Control Panel | Printers will be returned. If Default Properties is toggled on for the Default printer, this function will return a successful result, but the string handles will point to NULL strings.

C Syntax

```
BOOL CRPE_API PEGetSelectedPrinter (
        short printJob,
        HANDLE FAR *driverHandle,
        short FAR *driverLength,
        HANDLE FAR *printerHandle,
        short FAR *printerLength,
        HANDLE FAR *portHandle,
        short FAR *portLength,
    #if defined (WIN32)
        DEVMODEA FAR * FAR *mode
    #else
        DEVMODE FAR * FAR *mode
    #endif
        );
```

Parameters

printJob	Specifies the print job that you want to query to get information on the non-default printer that has been selected with the report.
driverHandle	Specifies a pointer to the handle of the printer driver for the printer that is selected with the print job.
driverLength	Specifies a pointer to the length of the printer driver name.
printerHandle	Specifies a pointer to the handle of the printer that is selected with the print job.
printerLength	Specifies a pointer to the length of the printer name.
portHandle	Specifies a pointer to the handle of the port to which the selected printer is connected.
portLength	Specifies a pointer to the length of the port name.
mode	Specifies a pointer to the “DEVMODE” on page 533, or DEVMODE Windows API structure.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

- Use “PEGetHandleString” on page 318, to obtain the actual strings pointed to by the string handles returned.

VB Syntax

```
Declare Function PEGetSelectedPrinter Lib "crpe32.dll" (
    ByVal printJob As Integer, DriverHandle As Long, DriverLength As
Integer,
    PrinterHandle As Long, PrinterLength As Integer, PortHandle As Long,
    PortLength As Integer, DevMode As Any ) As Integer
```

Delphi Syntax

```
function PEGetSelectedPrinter (
    printJob: Word;
    var driverHandle: Hwnd;
    var driverLength: Word;
    var printerHandle: Hwnd;
    var printerLength: Word;
    var portHandle: Hwnd;
    var portLength: Word;
    var mode: PDeviceModeA
    ): Bool stdcall;
```

PEGetSelectionFormula

Use PEGetSelectionFormula to retrieve the string handle for the selection formula used in the specified report. Use this function with “PEGetHandleString” on page 318. Use “PESetSelectionFormula” on page 441, to pass back the selection formula. The series can be used in a Custom-Print Link to identify and then change an existing record selection formula in response to a user selection at print time.

C Syntax

```
BOOL CRPE_API PEGetSelectionFormula (
        short printJob,
        HANDLE FAR *textHandle,
        short FAR *textLength );
```

Parameters

printJob	Specifies the print job for which you want to retrieve the selection formula string.
textHandle	Specifies a pointer to the handle of the string containing the formula text.
textLength	Specifies a pointer to the length of the formula string (in bytes) including the terminating byte.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEGetSelectionFormula Lib “crpe32.dll” (ByVal printJob
As Integer, TextHandle As Long, TextLength As Integer) As Integer
```

Delphi Syntax

```
function PEGetSelectionFormula (
    printJob: Word;
    var textHandle: HWnd;
    var textLength: Word
    ): Bool stdcall;
```

PEGetSQLExpression

Use PEGetSQLExpression to retrieve a specified SQL expression in the specified report. Use this function with "PEGetHandleString" on page 318.

C Syntax

```
BOOL CRPE_API PEGetSQLExpression (
        short printJob,
        const char FAR *expressionName,
        HANDLE FAR *textHandle,
        short FAR *textLength );
```

Parameters

printJob	Specifies the print job for which you want to retrieve the SQL expression string.
expressionName	Specifies a pointer to the expression name.
textHandle	Specifies a pointer to the handle of the expression string.
textLength	Specifies a pointer to the length of the expression string.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEGetSQLExpression Lib "crpe32.dll" (ByVal printJob As
Integer, ByVal expressionName As String, textHandle As Long, textLength
As Integer) As Integer
```

Delphi Syntax

```
function PEGetSQLExpression (
    printJob: Smallint;
    const expressionName: PChar;
    var textHandle: Hwnd;
    var textLength: Smallint
    ): Bool stdcall;
```

PEGetSQLQuery

Use PEGetSQLQuery to retrieve the same query that appears in the Show SQL Query dialog box in Crystal Reports, in a syntax that's specific to the database driver you're using. Use this function with "PEGetHandleString" on page 318. Use "PESetSQLQuery" on page 443, to update the query. See Remarks below.

C Syntax

```
BOOL CRPE_API PEGetSQLQuery (
        short printJob,
        HANDLE FAR *textHandle,
        short FAR *textLength );
```

Parameters

printJob	Specifies the print job from which you want to retrieve the SQL query.
textHandle	Specifies a pointer to the handle of the string containing the SQL query string.
textLength	Specifies a pointer to the length of the SQL query string (in bytes) including the terminating byte.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

The report must be logged on before the call to PEGetSQLQuery is made.

VB Syntax

```
Declare Function PEGetSQLQuery Lib "crpe32.dll" ( ByVal printJob As
Integer, TextHandle As Long, TextLength As Integer ) As Integer
```

Delphi Syntax

```
function PEGetSQLQuery (
    printJob: Word;
    var textHandle: HWnd;
    var textLength: Word
    ): Bool stdcall;
```

PEGetSubreportInfo

Use PEGetSubreportInfo to retrieve information about the specified subreport.

C Syntax

```
BOOL CRPE_API PEGetSubreportInfo (
        short printJob,
        DWORD subreportHandle,
        PESubreportInfo FAR *subreportInfo );
```

Parameters

printJob	Specifies the primary report that contains the subreport about which you want to retrieve information.
subreportHandle	Specifies the handle of the subreport about which you want to retrieve information.
subreportInfo	Specifies a pointer to “PESubreportInfo” on page 507, which will be used for holding the information once it is retrieved.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEGetSubreportInfo Lib “crpe32.dll” (
    ByVal printJob As Integer, ByVal subreportHandle As Long,
    subreportInfo As PESubreportInfo ) As Integer
```

Delphi Syntax

```
function PEGetSubreportInfo (
    printJob: Word;
    subreportHandle: DWORD;
    var subreportInfo: PESubreportInfo
    ): Bool stdcall;
```

PEGetTrackCursorInfo

Use PEGetTrackCursorInfo to track cursors. Different cursors can be specified for different report areas and report objects in the preview window. This function retrieves cursor information for a specified job.

C Syntax

```
BOOL CRPE_API PEGetTrackCursorInfo (
        short printJob,
        PETrackCursorInfo FAR *cursorInfo );
```

Parameters

printJob	Specifies the print job for which you want to retrieve track cursor information.
cursorInfo	Specifies a pointer to “PETrackCursorInfo” on page 514, which will contain the track cursor information.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Delphi Syntax

```
function PEGetTrackCursorInfo (
    printJob: smallint;
    var cursorInfo: PETrackCursorInfo
    ): Bool stdcall;
```

PEGetVersion

Use PEGetVersion to retrieve the version number of the DLL or the Crystal Report Engine. The high-order byte is the major version number and the low-order byte is the minor version number. This function can be used whenever you build functionality into a report that may not be available in earlier versions of the Crystal Report Engine and you need to verify the version number first. The function can be a handy safeguard for users who have more than one version of the Crystal Report Engine with the older version earlier in the path than the new version.

C Syntax

```
short CRPE_API PEGetVersion (
        short versionRequested );
```

Parameter

versionRequested	Specifies whether the DLL or Crystal Report Engine version is being requested. Use one of the following PE_GV_XXX constants.	
	Constant	**Description**
	PE_GV_DLL	Returns the version of the DLL (CRPE / CRPE32).
	PE_GV_ENGINE	Returns the version of the Crystal Report Engine.

Returns

Returns the version number of the DLL or the Crystal Report Engine.

VB Syntax

```
Declare Function PEGetVersion Lib "crpe32.dll" (ByVal version As
Integer)
    As Integer
```

The following simplifies the PEGetVersion call in Visual Basic.

```
Function PEVBGetVersion (ByVal component As Integer) As Single
    Dim version As Integer
    Dim major As Integer
    Dim minor As Integer
    version = PEGetVersion(component)
    If version = 0 Then
        PEVBGetVersion = 0
    Else
        major = version / 256
        minor = version Mod 256
        PEVBGetVersion = major + (minor / 10)
    End If
End Function
```

Delphi Syntax

```
function PEGetVersion (
    versionRequested: integer
    ): Smallint stdcall;
```

dBASE for Windows Syntax

```
EXTERN CWORD PEGetVersion (CWORD) CRPE.DLL
```

PEGetWindowHandle

Use PEGetWindowHandle to retrieve the handle of the preview window. This function can be used in a Custom-Print Link if you want to do something with the preview window (move it, change its size, etc.). PEGetWindowHandle can also be used to determine if the user has already closed the preview window.

C Syntax

```
HWND CRPE_API PEGetWindowHandle (
        short printJob );
```

Parameter

printJob	Specifies the print job for which you want to retrieve the preview window handle. If two or more preview windows are open, this function applies only to the most recently created preview window.

Returns

- Returns the preview window handle if the call is successful.
- Returns 0 if an error occurs or if the preview window has already been closed.

Remarks

This function can be used after “PEStartPrintJob” on page 448, and then, only if you have created a preview window.

VB Syntax

```
Declare Function PEGetWindowHandle Lib “crpe32.dll” (
    ByVal printJob As Integer ) As Long
```

Delphi Syntax

```
function PEGetWindowHandle (
    pr6intJob: Word
    ): HWnd stdcall;
```

dBASE for Windows Syntax

```
EXTERN CHANDLE PEGetWindowHandle (CWORD) CRPE.DLL
```

PEGetWindowOptions

Use PEGetWindowOptions to configure the preview window look and functionality. You can also determine whether the preview window has a group tree window; whether it can be drill down if there are hidden groups; and whether the close button, refresh button, and print setup button are shown, for example. This function returns the current report preview window configuration.

C Syntax

```
BOOL CRPE_API PEGetWindowOptions (
        short printJob,
        PEWindowOptions FAR *options );
```

Parameters

printJob	Specifies the print job from which you want to retrieve the preview window options. If two or more preview windows are open, this function applies only to the most recently created preview window.
PEWindowOptions	Specifies a pointer to “PEWindowOptions” on page 519, which will contain the retrieved information.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

When hasGroupTree is True, it does not mean there will be a group tree in the preview window. The hasGroupTree option and the report option (Create Group Tree in Crystal Reports) together determine the group tree visibility in the preview window. Both options must be True for the group tree to be shown.

VB Syntax

```
Declare Function PEGetWindowOptions Lib “crpe32.dll” (
    ByVal printJob As Integer, Options As PEWindowOptions ) As Integer
```

Delphi Syntax

```
function PEGetWindowOptions (
    printJob: Word;
    var options: PEWindowOptions
    ):Bool stdcall;
```

PEHasSavedData

Use PEHasSavedData to determine if the specified report has data saved with it in memory. With this information, you can determine whether or not the data needs to be refreshed before the report is printed. See Remarks below.

C Syntax

```
BOOL CRPE_API PEHasSavedData (
        short printJob,
        BOOL FAR *hasSavedData );
```

Parameters

printJob	Specifies the handle of the print job you want to query to determine if it has saved data with it.
hasSavedData	Specifies a pointer to a Boolean value that indicates whether or not there is data saved with the report.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

- A report may or may not have saved data when a print job is first opened from a report file. Since data is saved during a print, however, a report will always have saved data immediately after it is printed.
- The default behavior is for a report to use its saved data rather than refresh its data from the database when printing a report.
- Use “PEDiscardSavedData” on page 295, to release the saved data associated with a report. The next time the report is printed, it will get current data from the database.

VB Syntax

```
Declare Function PEHasSavedData Lib “crpe32.dll” (
    ByVal printJob As Integer, HasSavedData As Long ) As Integer
```

Delphi Syntax

```
function PEHasSavedData(
    printJob: Word;
    var hasSavedData: Bool
    ): Bool stdcall;
```

PEIsPrintJobFinished

Use PEIsPrintJobFinished to monitor the print job to see if it is finished or still in progress. You can use this function any time you have a call that is contingent on a print job being finished.

C Syntax

```
BOOL CRPE_API PEIsPrintJobFinished (
        short printJob );
```

Parameter

printJob	Specifies the print job that you want to query to determine if it has finished printing.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

PEIsPrintJobFinished will return TRUE immediately after the report has been displayed in the preview window, even if that preview window is still open.

VB Syntax

```
Declare Function PEIsPrintJobFinished Lib "crpe32.dll" (ByVal printJob _
        As Integer) As Integer
```

Delphi Syntax

```
function PEIsPrintJobFinished (
    printJob: Word
    ): Bool stdcall;
```

dBASE for Windows Syntax

```
EXTERN CLOGICAL PEIsPrintJobFinished (CWORD) CRPE.DLL
```

PELogOffServer

Use PELogOffServer to Log off the specified server. Use this call any time you have to log off a specified server.

C Syntax

```
BOOL CRPE_API PELogOffServer (
        const char FAR *dllName,
        PELogOnInfo FAR *logOnInfo );
```

Parameters

dllName	Specifies a pointer to the name of the Crystal Reports DLL for the datasource from which you want to log off (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 and PDS*.DLL for SQL / ODBC databases.
logOnInfo	Specifies a pointer to "PELogOnInfo" on page 479.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

- "PELogOnServer" on page 374, and PELogOffServer can be called at any time to log on and off a database server. These functions are not required if the function "PESetNthTableLogOnInfo" on page 427, was already used to logon to a table.
- This function requires a database DLL name which can be retrieved using "PEGetNthTableType" on page 350.

VB Syntax

```
Declare Function PELogOffServer Lib "crpe32.dll" ( ByVal DLLName As
String, LogOnInfo As PELogOnInfo ) As Integer
```

Delphi Syntax

```
function PELogOffServer(
    dllName: PChar;
    var logOnInfo: PELogOnInfo
    ): Bool stdcall;
```

PELogOnServer

Use PELogOnServer to logon to the specified server.

C Syntax

```
BOOL CRPE_API PELogOnServer (
        const char FAR *dllName,
        PELogOnInfo FAR *logOnInfo );
```

Parameters

dllName	Specifies a pointer to the name of the Crystal Reports DLL for the server or password protected non-SQL table to which you want to logon. (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 and PDS*.DLL for SQL / ODBC databases.
logOnInfo	Specifies a pointer to "PELogOnInfo" on page 479.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

- PELogOnServer and “PELogOffServer” on page 373, can be called at any time to log on and off a database server. These functions are not required if “PESetNthTableLogOnInfo” on page 427, was already used to set the password for a table.
- This function requires a database DLL name, which can be retrieved using “PEGetNthTableType” on page 350.
- This function can also be used for non-SQL tables, such as password-protected Paradox tables. Call this function to set the password for the Paradox DLL before beginning printing.
- When printing using “PEStartPrintJob” on page 448, the ServerName passed in PELogOnServer does not need to agree exactly with the server name stored in the report. PELogOnServer can be used to switch to a different server or datasource at runtime.
- The following points need to be considered when deciding whether to use PELogOnServer or PESetNthTableLogOnInfo.
  - PELogOnServer is easier to call than “PESetNthTableLogOnInfo” on page 427, and it can be called at any time. However, you must know the database DLL name to make this call.
  - PESetNthTableLogOnInfo is more flexible than PELogOnServer. It allows you to override any of the logon parameters. PESetNthTableLogOnInfo must be called after “PEOpenPrintJob” on page 378.

VB Syntax

```
Declare Function PELogOnServer Lib "crpe32.dll" ( ByVal DLLName As
String, LogOnInfo As PELogOnInfo ) As Integer
```

Delphi Syntax

```
function PELogOnServer(
    dllName: PChar;
    var logOnInfo: PELogOnInfo
    ): Bool stdcall;
```

PELogOnSQLServerWithPrivateInfo

Use PELogOnSQLServerWithPrivateInfo to enable the Crystal Report Engine to “piggyback” your application’s existing connection to a Server. If you are already logged on, this function lowers the number of connections established by a workstation, thus reducing application time and network traffic. It also prevents a Crystal Reports Log Off call from disconnecting an application’s existing connection to the Server.

C Syntax

```
BOOL CRPE_API PELogOnSQLServerWithPrivateInfo (
        const char FAR *dllName,
        void FAR *privateInfo );
```

Parameters

dllName	Specifies a pointer to the name of the Crystal Reports DLL that was used in establishing the connection to the server when the report was first created. For example, if a report was created using an ODBC datasource, the Crystal Reports DLL is PDSODBC.DLL.
privateInfo	In the application, a connection to the server has to have been established and this in turn generates a Handle to a Database Connection (HDBC). This parameter specifies the application's handle to the connection. This makes Crystal Reports aware of the existing connection so it can use it instead of establishing a new one. Since the reports with which this function works are based on ODBC, this parameter is actually an ODBC HDBC.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

If the application uses ODBC to connect, get the ODBC HDBC by using the following function calls. Also, see the ODBC documentation for more information.

```
SQLAllocEnv
«Initializes the ODBC call level interface and allocates memory for an
environment handle.»
SQLAllocConnect
«Returns an ODBC HDBC.»
```

VB Syntax

```
Declare Function PELogOnSQLServerWithPrivateInfo Lib "crpe32.dll" (
ByVal DLLName As String, ByVal PrivateInfo As Long ) As Integer
```

Delphi Syntax

```
function PELogOnSQLServerWithPrivateInfo (
    dllName: PChar;
    privateInfo: Pointer
    ): Bool stdcall;
```

PENextPrintWindowMagnification

Use PENextPrintWindowMagnification to change the preview window magnification to the next magnification level in sequence. Use this function to cycle through the three levels of preview window magnification (Full Page, Fit One Side, Fit Both Sides, Full Page, Fit One Side, etc.) whenever the report has been printed to a preview window.

C Syntax

```
BOOL CRPE_API PENextPrintWindowMagnification (
        short printJob );
```

Parameter

printJob	Specifies the print job displayed in the preview window for which you want to step the magnification level.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PENextPrintWindowMagnification Lib "crpe32.dll" (ByVal
    printJob As Integer ) As Integer
```

Delphi Syntax

```
function PENextPrintWindowMagnification (
    printJob: Word
    ): Bool stdcall;
```

dBASE for Windows Syntax

```
EXTERN CLOGICAL PENextPrintWindowMagnification (CWORD) CRPE.DLL
```

PEOpenEngine

Use PEOpenEngine to prepare the Crystal Report Engine (in single-thread mode) for requests. This function is a necessary part of any Custom-Print Link. It is also required for any Print-Only Link in which you want the report to print to a window that is to remain visible after the report is printed. It is not necessary to use this function with a Print-Only Link if the report is directed to a printer.

C Syntax

```
BOOL CRPE_API PEOpenEngine (void);
```

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

- This function must be called before any other Crystal Report Engine function. If an error occurs in the PEOpenEngine function call, “PEGetErrorCode” on page 304, can be passed a print job value of zero to obtain error information.
- PEOpenEngine opens the print engine in single-thread mode by default.

VB Syntax

```
Declare Function PEOpenEngine Lib “crpe32.dll” () As Integer
```

Delphi Syntax

```
function PEOpenEngine
    ): Bool stdcall;
```

dBASE for Windows Syntax

```
EXTERN CLOGICAL PEOpenEngineEx () CRPE.DLL
```

PEOpenPrintJob

Use PEOpenPrintJob to prepare to print a report and return a handle which identifies that particular print job. The handle returned must be used in all subsequent calls related to that print job which require that information. This function is used as a mandatory part of a Custom-Print Link to retrieve the print job handle which is then used when required as parameter printJob in each additional Custom-Print Link function call.

C Syntax

```
short PEOpenPrintJob (
        const char *reportFilePath );
```

Parameter

reportFilePath	Specifies the file name and path of the report that you want to open. You must enclose this parameter in quotes.

Returns

- Returns the job number.
- Returns 0 if the report file does not exist or if an error occurs.

Remarks

- This function must be called before most other Crystal Report Engine functions are used.
- Only one print job can be configured at a time.
- "PEClosePrintJob" on page 288, must be called later to close the job.
- Report Path\Filename must be enclosed in quotes.
  - For example, PEOpenPrintJob ("C:\CRW\REPORT1.RPT");
  - Note: In C or C++, the slashes (\) in the string must be entered as double slashes (\\).
- This function opens the print job with the printer selected in the report (via the File | Printer Setup menu command) or the default printer (if no replacement printer has been selected in the report).

VB Syntax

```
Declare Function PEOpenPrintJob Lib "crpe32.dll" (ByVal RptName As
String) _
        As Integer
```

Delphi Syntax

```
function PEOpenPrintJob (
    reportFilePath: PChar
    ): Smallint stdcall;
```

dBASE for Windows Syntax

```
EXTERN CWORD PEOpenPrintJob (CSTRING) CRPE.DLL
```

PEOpenSubreport

Use PEOpenSubreport to open the named subreport and return a number that identifies that subreport.
The number returned must be used in all subsequent calls related to the subreport (where a print job handle is required).

C Syntax

```
short CRPE_API PEOpenSubreport (
        short parentJob,
        char FAR *subreportName );
```

Parameters

parentJob	Specifies the primary report (the report that contains the subreport). This is the handle returned from PEOpenPrintJob.
subreportName	Specifies a pointer to the name of the subreport that you want to open. This is retrieved using "PEGetSubreportInfo" on page 367.

Returns

- Returns the job number of the subreport.
- Returns 0 if the subreport does not exist or if an error occurs.

Remarks

- This function must be called before any other Crystal Report Engine functions related to the subreport.
- PECloseSubreport must be called later to close the job.

VB Syntax

```
Declare Function PEOpenSubreport Lib "crpe32.dll" (
    ByVal parentJob As Integer, ByVal subreportName As String ) As
Integer
```

Delphi Syntax

```
function PEOpenSubreport (
    parentJob: Word;
    subreportName: PChar
    ): Word stdcall;
```

PEOutputToPrinter

Use PEOutputToPrinter to direct output to a printer. See Remarks below.

C Syntax

```
BOOL CRPE_API PEOutputToPrinter (
        short printJob,
        short nCopies );
```

Parameters

printJob	Specifies the print job that you want to send to a printer.
nCopies	Specifies the number of report copies that you want to print. Pass 0 to preserve the existing setting.

Returns

- TRUE if the output can be sent to the printer successfully.
- FALSE if the output cannot be sent to the printer.

Remarks

- If a printer has been specified via "PESelectPrinter" on page 389, output will be sent to that printer.
- If there is no PESelectPrinter selection but there is a printer specified in the report via the Print | Select Printer menu command, output will be sent to that printer.

- If there is no PESelectPrinter selection and there is no printer specified in the report, output will be to the Windows default printer.
- "PEOpenPrintJob" on page 378, opens the print job with the printer specified in the report (if there is one) or with the Windows default printer (if no printer is specified in the report).
- The sequence of calls that follows may help to explain printer output concepts as well as potential problems. Assume that a printer is specified in the report via the Print | Select Printer menu command. Make certain to sequence your function calls to get the output desired.
  - PEOpenPrintJob

```
// Opens the job with printer specified in report, or, if none
// is specified, the Windows default printer. The printer the
// job opens with is Printer #1.
```

  - PEOutputToWindow

```
// Directs the output to the preview window.
```

  - PEStartPrintJob

```
// Report is printed in the preview window based on Printer #1.
```

  - PEOutputToPrinter

```
// Directs output to the printer.
```

  - PESelectPrinter

```
// Specifies 2nd printer, Printer #2. This overrides Printer #1.
```

  - PEStartPrintJob

```
// Report is printed on Printer #2. Window output and printer
// output are based on two different printers and may cause
// confusion.
```

  - PEClosePrintJob
- If one printer is set for landscape output, for example, and the other for portrait output, the sequence of calls above will print an entirely different report in the preview window than what actually appears on paper.
- This function supersedes PEOutputToDefaultPrinter which was available in earlier versions of the Crystal Report Engine.

VB Syntax

```
Declare Function PEOutputToPrinter Lib "crpe32.dll" (
    ByVal printJob As Integer, ByVal nCopies As Integer ) As Integer
```

Delphi Syntax

```
function PEOutputToPrinter (
    printJob: Word;
    nCopies: integer
    ): Bool stdcall;
```

dBASE for Windows Syntax

```
EXTERN CLOGICAL PEOutputToPrinter (CWORD, CWORD) CRPE.DLL
```

PEOutputToWindow

Use PEOutputToWindow to direct printed output to a preview window. This function is used as part of a Custom-Print Link whenever you want the report printed to the preview window instead of to the printer.

C Syntax

```
BOOL CRPE_API PEOutputToWindow (
        short printJob,
        const char FAR *title,
        int left,
        int top,
        int width,
        int height,
        DWORD style,
        HWND parentWindow );
```

Parameters

printJob	Specifies the print job you want to print in the preview window.
title	Specifies a pointer to the null-terminated string that contains the title that you want to appear in the preview window title bar.
left	Specifies the x coordinate of the upper left corner of the preview window, in pixels. See Remarks below.
top	Specifies the y coordinate of the upper left corner of the preview window, in pixels. See Remarks below.
width	Specifies the width of the preview window, in pixels.
height	Specifies the height of the preview window, in pixels.
style	Specifies the style of the window being created. Style settings can be combined using the bitwise “OR” operator. You can specify any of the following window styles. Also, see Remarks below.

Constant	Value	Description
WS_MINIMIZE	536870912	Make a window of minimum size.
WS_VISIBLE	268435456	Make a window that is visible when it first appears (for overlapping and pop-up windows).
WS_DISABLED	134217728	Make a window that is disabled when it first appears.
WS_CLIPSIBLINGS	67108864	Clip child windows with respect to one another.
WS_CLIPCHILDREN	33554432	Exclude the area occupied by child windows when drawing inside the parent window.
WS_MAXIMIZE	16777216	Make a window of maximum size.
WS_CAPTION	12582912	Make a window that includes a title bar.
WS_BORDER	8388608	Make a window that includes a border.

	WS_DLGFRAME	4194304	Make a window that has a double border but no title.
	WS_VSCROLL	2097152	Make a window that includes a vertical scroll bar.
	WS_HSCROLL	1048576	Make a window that includes a horizontal scroll bar.
	WS_SYSMENU	524288	Include the system menu box.
	WS_THICKFRAME	262144	Include the thick frame that can be used to size the window.
	WS_MINIMIZEBOX	131072	Include the minimize box.
	WS_MAXIMIZEBOX	65536	Include the maximize box.
	CW_USEDFAULT	-32768	Assign the child window the default horizontal and vertical position, and the default height and width.
parentWindow	Specifies the handle of the parentWindow if the preview window is a child of that window.		

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

- For a top-level preview window, the top left corner is relative to the origin of the screen. For an MDI child preview window, the top left corner is relative to the origin of the frame window's client area. For a child preview window, the top left corner is relative to the origin of the parent window's client area.
- If *parentWindow* is NULL, the preview window is a top-level window (that is, not a child of any other window). For a top-level preview window, the top left corner is relative to the origin of the screen. *Left* and *top* can be CW_USDEFAULT to put the window at a default location. *Width* and *height* can also be CW_USEDEFAULT to give the window a default size.
- If the MDI frame window parent handle is specified, the report preview window will show up in the client area of the MDI parent. If the MDI frame window child window handle is specified (child window must be created) the report preview window will show up in the child window.
- If parentWindow is the handle of some other window, the preview window is a child of that window. For a child preview window, the top left corner is relative to the origin of the parent window's client area.
- If the preview window is a top-level window or an MDI child window and *style* is 0, the following style is used instead.

```
(WS_VISIBLE | WS_THICKFRAME | WS_SYSMENU | WS_MAXIMIZEBOX |
WS_MINIMIZEBOX)
```

- That is, the default window is a visible window with a thick frame that can be used for sizing the window. The window includes a system menu box, and a maximize and minimize box.

- The preview window is created when "PEStartPrintJob" on page 448, is called.

VB Syntax

```
Declare Function PEOutputToWindow Lib "crpe32.dll" (ByVal printJob As
Integer, ByVal Title As String, ByVal Left As Long, ByVal Top As Long,
ByVal Width As Long, ByVal Height As Long, ByVal style As Long, ByVal
PWindow As Long) As Integer
```

Visual Basic developers can cut and paste a declaration for CW_USEDEFAULT into their application.

- For VB 4, cut the declaration from c:\vb\winapi\win32api.txt
- For VB 3, cut the declaration from c:\vb\winapi\win30api.txt

Delphi Syntax

```
function PEOutputToWindow (
    printJob: Word;
    title: PChar;
    left: longint;
    top: longint;
    width: longint;
    height: longint;
    style: longint;
    parentWindow: HWnd
    ): Bool stdcall;
```

dBASE for Windows Syntax

```
EXTERN CLOGICAL PEOutputToWindow (CWORD, CSTRING, CWORD, CWORD, CWORD,
CWORD, CLONG, CHANDLE) CRPE.DLL
```

PEPrintControlsShowing

Use PEPrintControlsShowing to determine if the print controls are displayed in the preview window. Use this function to retrieve the visible print controls and pass back using "PEShowPrintControls" on page 447.

C Syntax

```
BOOL CRPE_API PEPrintControlsShowing (
        short printJob,
        BOOL FAR *controlsShowing );
```

Parameters

printJob	Specifies the print job for which you want to determine whether or not the print controls will be displayed when the job is sent to a preview window.
controlsShowing	Specifies a pointer to a TRUE value if the print controls will be shown or FALSE value if they will be hidden.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEPrintControlsShowing Lib "crpe32.dll" (
    ByVal printJob As Integer, ControlsShowing As Long ) As Integer
```

Delphi Syntax

```
function PEPrintControlsShowing (
    printJob: Word;
    var controlsShowing: Bool
    ): Bool stdcall;
```

PEPrintReport

Use PEPrintReport to print the specified report to either the printer or to a preview window. This function establishes a Print-Only Link where changes made during runtime by other PE calls are ignored. Use PEPrintReport any time that you simply want to print a report from an application without giving the user the ability to customize the report.

C Syntax

```
short CRPE_API PEPrintReport (
        const char FAR *reportFilePath,
        BOOL toDefaultPrinter,
        BOOL toWindow,
        const char FAR *title,
        int left,
        int top,
        int width,
        int height,
        DWORD style,
        HWND parentWindow );
```

Parameters

reportFilePath	Specifies a pointer to the NULL-terminated string that contains the name and path of the report that you want to print.
toDefaultPrinter	Specifies whether or not the report is to be sent to the default printer.
toWindow	Specifies whether or not the report is to be displayed in the preview window.
title	Specifies a pointer to the NULL-terminated string that contains the title you want to appear on the title bar if you are printing the report to a window.
left	Specifies the x coordinate of the upper left hand corner of the preview window, in pixels.
top	Specifies the y coordinate of the upper left hand corner of the preview window, in pixels.
width	Specifies the width of the preview window, in pixels.
height	Specifies the height of the preview window, in pixels.
style	Specifies the style of the window being created. Style settings can be combined using the bitwise "OR" operator. Select your style from the list that appears with "PEOutputToWindow" on page 382.
parentWindow	Specifies the handle of the parent window if the preview window is a child of that window.

Returns

- Returns PE_ERR_NOERROR if the call was successful.
- Returns another "Error Codes" on page 545, if the call failed.

VB Syntax

```
Declare Function PEPrintReport Lib "crpe32.dll" (ByVal RptName As
String, ByVal Printer As Integer, ByVal Window As Integer, ByVal Title As
String, ByVal Lft As Long, ByVal Top As Long, ByVal Wdth As Long, ByVal
Height As Long, ByVal Style As Long, ByVal PWindow As Long) As Integer
```

Delphi Syntax

```
function PEPrintReport (
    reportFilePath: PChar;
    toDefaultPrinter: Bool;
    toWindow: Bool;
    title: PChar;
    left: integer;
    top: integer;
    width: integer;
    height: integer;
    style: longint;
    parentWindow: HWnd
    ): Smallint stdcall;
```

dBASE for Windows Syntax

```
EXTERN CWORD PEPrintReport (CSTRING, CLOGICAL, CLOGICAL, CSTRING, CWORD,
CWORD, CWORD, CWORD, CLONG, CHANDLE) CRPE.DLL
```

PEPrintWindow

Use PEPrintWindow to print the report that is displayed in the preview window. This function can be used in a Custom-Print Link to enable the user to preview the report in the preview window, and then, if everything looks satisfactory, to print the report to the printer in response to a user event.

C Syntax

```
BOOL CRPE_API PEPrintWindow (
        short printJob,
        BOOL waitUntilDone );
```

Parameters

printJob	Specifies the print job displayed in the preview window that you want to print.
waitUntilDone	BOOL. Reserved. This parameter must always be set to TRUE.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEPrintWindow Lib "crpe32.dll" (
    ByVal printJob As Integer, ByVal WaitNoWait As Integer ) As Integer
```

Delphi Syntax

```
function PEPrintWindow (
    printJob: Word;
    waitUntilDone: Bool
    ): Bool stdcall;
```

dBASE for Windows Syntax

```
EXTERN CLOGICAL PEPrintWindow (CWORD, CLOGICAL) CRPE.DLL
```

PEReimportSubreport

Use PEReimportSubreport to reimport a subreport into the specified main report.

C Syntax

```
BOOL CRPE_API PEReimportSubreport (
    short printJob,
    DWORD subreportHandle,
    BOOL FAR *linkChanged,
    BOOL FAR *reimported );
```

Parameters

printJob	Specifies the print job for which you want to reimport a subreport.
subreportHandle	Specifies the handle of the subreport that you want to reimport.
linkChanged	Specifies a pointer to a Boolean value indicating whether or not the link has changed. See Remarks below.
reimported	Specifies a pointer to a Boolean value indicating whether or not the subreport has been reimported. See Remarks below.

Returns

- TRUE if the call is successful and the subreport is up-to-date or was reimported with fixed or missing links.
- FALSE if the call fails, the subreport path is invalid, or the reimport failed. The specific error that occurred can be retrieved with “PEGetErrorCode” on page 304.

Remarks

- Parameter linkChanged
  - will be set to FALSE if the subreport is reloaded and the links are fixed.
  - will be set to TRUE if the subreport is reloaded but missing links.
- Parameter reimported
  - will be set to FALSE if the subreport is up-to-date, or if the reimport failed due to an invalid path or other error.
  - will be set to TRUE if the subreport was reloaded with links fixed or missing.

VB Syntax

```
Declare Function PEReimportSubreport Lib "crpe32.dll" (
    ByVal printJob As Integer, ByVal subreportHandle As Long,
    linkChanged As Long, reimported As Long ) As Integer
```

PESelectPrinter

Use PESelectPrinter to specify a printer other than the default printer as the print destination for the specified print job. You can use this function to enable the user to select a printer other than the default printer at print time. One way of doing this is to have your application call the Print Setup common dialog box.

C Syntax

```
BOOL CRPE_API PESelectPrinter (
        short printJob,
        const char FAR *driverName,
        const char FAR *printerName,
        const char FAR *portName,
        DEVMODEA FAR *mode );
```

Parameters

printJob	Specifies the print job for which you want to select a printer.
driverName	Specifies a pointer to a null-terminated string that contains the name of the printer driver for the selected printer.
printerName	Specifies a pointer to a null-terminated string that contains the printer name for the selected printer.
portName	Specifies a pointer to a null-terminated string that contains the port name for the port to which the selected printer is attached.
mode	Specifies a pointer to the “DEVMODE” on page 533, Windows API structure.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

- The PRINTDLG structure returned by the Windows API PrintDlg function contains handles to DEVMODEA and DEVNAMES structures. This information can be used to obtain driverName, printerName, portName, and mode values for PESelectPrinter.
- Your code must parse the return from the dialog box selection and insert the returned Printer Driver Name, Printer Name, and Port Name as parameters in the call.
- After selecting the printer with this call, you can direct the output to that printer (using “PEOutputToPrinter” on page 380) or to the preview window (using “PEOutputToWindow” on page 382).
- This call will override a printer selection that you built into the report at design time via the Crystal Reports Select Printer menu command.

- If you follow this call with the call “PEOutputToWindow” on page 382, the report appears in the preview window.
- To revert to the default printer, pass 0 for each parameter.
- The driver name and printer name must exist on your machine.
- You can specify a different printer port than that assigned to the selected printer on your machine.
- For parameter mode, use 0 for the default mode or create a “DEVMODE” on page 533, structure to customize (if your development tool supports such a structure).
- This function should be called before “PEStartPrintJob” on page 448, or the results may be inconsistent or unexpected.

VB Syntax

```
Declare Function PESelectPrinter Lib "crpe32.dll" ( ByVal printJob As
Integer,
    ByVal PrinterDriver As String, ByVal PrinterName As String,
    ByVal PortName As String, DevMode As Any ) As Integer
```

Delphi Syntax

```
function PESelectPrinter(
    printJob: Word;
    driverName: PChar;
    printerName: PChar;
    portName: PChar;
    mode: PDeviceModeA
    ): Bool stdcall;
```

PESetAllowPromptDialog

Use PESetAllowPromptDialogto specify whether prompting for parameter values is allowed during printing.

C Syntax

```
BOOL CRPE_API PESetAllowPromptDialog (
        short printJob,
        BOOL showPromptDialog );
```

Parameters

printJob	Specifies the print job.
showPromptDialog	If TRUE, then prompting is allowed.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PESetAllowPromptDialog Lib "crpe32.dll" (ByVal _
    printJob As Integer, ByVal showPromptDialog As Integer) As Integer
```

Delphi Syntax

```
function PESetAllowPromptDialog (
    printJob: Smallint;
    showPromptDialog: Bool
    ): Bool stdcall;
```

PESetAreaFormat

Use PESetAreaFormat to set the area format settings for selected areas in the specified report to the values in "PESectionOptions" on page 501. This function can be used to provide specialized formatting for printing invoices, form letters, printing to pre-printed forms, etc. It allows you to hide an area, insert a page break either before or after an area begins, reset the page number to 1 after a group value prints, prevent page breaks from spreading data from a single record over two pages, and to print group values only at the bottom of a page.

C Syntax

```
BOOL CRPE_API PESetAreaFormat (
        short printJob,
        short areaCode,
        PESectionOptions FAR *options );
```

Parameters

printJob	Specifies the print job for which you want to set area formatting options.
areaCode	Specifies the "Section Codes" on page 559 for the report area for which you want to set formatting options. See the information on area codes in "Working with section codes" on page 46.
options	Specifies a pointer to "PESectionOptions" on page 501. Use this structure to set your section options.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

This function should be called before "PEStartPrintJob" on page 448, or the results may be inconsistent or unexpected.

VB Syntax

```
Declare Function PESetAreaFormat Lib "crpe32.dll" ( ByVal printJob As
Integer,
    ByVal areaCode As Integer, Options As PESectionOptions ) As Integer
```

Delphi Syntax

```
function PESetAreaFormat (
    printJob: Word;
    areaCode: Integer;
    var options: PESectionOptions
    ): Bool stdcall;
```

PESetAreaFormatFormula

Use PESetAreaFormatFormula to change the specified area format formula to the formula string you supply as a parameter. This function will only change the text of a formula which already exists in the report; you cannot use it to add a formula. When you give the user the ability to change the formula at print time, your link must include code to replace the formulaString parameter with a user-generated value.

C Syntax

```
BOOL CRPE_API PESetAreaFormatFormula (
        short printJob,
        short areaCode,
        short formulaName,
        const char FAR *formulaString );
```

Parameters

printJob	Specifies the print job for which you want to set a new format formula string.
areaCode	Specifies the "Section Codes" on page 559 for the report area for which you want to set formatting options. See "Working with section codes" on page 46.
formulaName	Specifies the name of the formatting formula for which you want to supply a new string. Use one of the PE_FFN_XXX "Area/Section Format Formula Constants" on page 541.
formulaString	Specifies a pointer to the null-terminated string that you want to assign to the format formula.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.
- Error code PE_ERR_BADFORMULANAME if the formula does not exist.

- Error code PE_ERR_BADFORMULATEXT if there is an error in the formula.

Remarks

- This function should be called before "PEStartPrintJob" on page 448, or the results may be inconsistent or unexpected.
- Not all parameters apply to all areas.

VB Syntax

```
Declare Function PESetAreaFormatFormula Lib "crpe32.dll" (
    ByVal printJob As Integer, ByVal areaCode As Integer,
    ByVal formulaName As Integer, ByVal formulaString As String ) As
Integer
```

Delphi Syntax

```
function PESetAreaFormatFormula(
    printJob: Word;
    areaCode: Word;
    formulaName: Word;
    formulaString: Pchar
    ): Bool stdcall;
```

PESetDialogParentWindow

Use PESetDialogParentWindow to set the handle for the parent window of CRPE dialog boxes (that is, Print Progress dialog box).

C Syntax

```
BOOL CRPE_API PESetDialogParentWindow (
        short printJob,
        HWND parentWindow );
```

Parameters

printJob	Specifies the print job for which you want to specify a parent window.
parentWindow	Specifies the handle of the parent window.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PESetDialogParentWindow Lib "crpe32.dll" (ByVal
printJob _
        As Integer, ByVal parentWindow As Long) As Integer
```

Delphi Syntax

```
function PESetDialogParentWindow (
    printJob: Word;
    parentWindow: HWnd
    ): Bool stdcall;
```

PESetEventCallback

Use PESetEventCallback to set the event callback function for the specified job. CRPE can fire certain events when something happens inside CRPE. CRPE will call the callback function and notify what kind of event has or is about to occur. Within *callbackProc*, the user can interpret the event ID and perform the proper process.

C Syntax

```
BOOL CRPE_API PESetEventCallback (
        short printJob,
        BOOL (CALLBACK *callbackProc)
            (short eventID, void *param, void *userData)
        void *userData );
```

Parameters

printJob	Specifies the print job for which you want to create an Event callback procedure.
callbackProc	The CALLBACK procedure that will handle your Crystal Report Engine events. This should be a pointer to a standard Windows CALLBACK procedure. Refer to the Windows SDK for information on creating CALLBACK procedures.
userData	Specifies a pointer to any information you want to pass to the Event handling CALLBACK procedure. The pointer will be available in the userData member of the procedure. This value can be 0.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

- Each job can have only one callback function.
- The Event procedure functions passed in the *callbackProc* parameter should be a standard Windows CALLBACK procedure. Refer to documentation on the Windows API for information on creating CALLBACK procedures.
- If you need to pass data to *callbackProc* using the userData parameter of *callbackProc*, be sure the memory allocated for the data does not fall out of scope or gets deallocated before the *callbackProc* is called by Windows. If this happens, the data will be unavailable and errors may occur in your application.
- For a complete example of how to use this function, see "Handling preview window events" on page 59.

- If *callbackProc* returns TRUE, a CRPE default action will be provided. If *callbackProc* returns FALSE, a CRPE default action will not be used. The user should be responsible for providing default behavior. For some events, the *callbackProc* return value is ignored. The following list gives the description of different events supported by CRPE.

PE_ACTIVATE_PRINT_WINDOW_EVENT

- **Called**
  Before the preview window becomes active.
- **Parameter**
  Pointer to "PEGeneralPrintWindowEventInfo" on page 467.
- **Return**
  Ignored.

PE_CANCEL_BUTTON_CLICKED_EVENT

- **Called**
  After clicking the cancel button; before canceling the printing or reading the database.
- **Parameter**
  Pointer to "PEGeneralPrintWindowEventInfo" on page 467.
- **Return**
  TRUE to use the default action; FALSE to cancel the default action.

PE_CLOSE_BUTTON_CLICKED_EVENT

- **Called**
  After clicking the close button; before closing the preview window.
- **Parameter**
  Pointer to "PECloseButtonClickedEventInfo" on page 454.
- **Return**
  TRUE to use the default action; FALSE to cancel the default action.
- **Remarks**
  If FALSE is returned before actually closing the preview window, a PE_CLOSE_PRINT_WINDOW_EVENT is fired.

PE_CLOSE_PRINT_WINDOW_EVENT

- **Called**
  Before the preview window is closed.
- **Parameter**
  Pointer to "PEGeneralPrintWindowEventInfo" on page 467.
- **Return**
  TRUE to use the default action; FALSE to cancel the default action.

PE_DEACTIVATE_PRINT_WINDOW_EVENT

- **Called**
  Before the preview window becomes in active.
- **Parameter**
  Pointer to "PEGeneralPrintWindowEventInfo" on page 467.
- **Return**
  Ignored.

PE_DRILL_ON_DETAIL_EVENT

- **Called**
  After double-clicking one of the detail areas in the preview window.
- **Parameter**
  Pointer to "PEDrillOnDetailEventInfo" on page 454.
- **Return**
  Ignored.

PE_DRILL_ON_GROUP_EVENT

- **Called**
  After clicking on one of the group tree nodes, double-clicking or Ctrl-clicking a node with the magnify glass cursor, or double-clicking one of the groups in the preview window; before showing the group. This event also applies to drilling on a graph.
- **Parameter**
  Pointer to "PEDrillOnGroupEventInfo" on page 455.
- **Return**
  TRUE to use the default action; FALSE to cancel the default action.

PE_DRILL_ON_HYPERLINK_EVENT

- **Called**
  Whenever the user double-clicked on an object with a hyperlink (and the hyperlink is about to be executed).
- **Parameter**
  Pointer to"PEHyperlinkEventInfo" on page 477.
- **Return**
  TRUE to use the default action; FALSE to cancel the default action.

PE_EXPORT_BUTTON_CLICKED_EVENT

- **Called**
  After Export button is clicked; before export process starts.
- **Parameter**
  Pointer to "PEGeneralPrintWindowEventInfo" on page 467.
- **Return**
  TRUE to use the default action; FALSE to cancel the default action.

PE_FIRST_PAGE_BUTTON_CLICKED_EVENT

- **Called**
  After clicking the first page button; before going to the first page.
- **Parameter**
  Pointer to "PEGeneralPrintWindowEventInfo" on page 467.
- **Return**
  TRUE to use the default action; FALSE to cancel the default action.

PE_GROUP_TREE_BUTTON_CLICKED_EVENT

- **Called**
  After clicking the group tree button; before showing or hiding the group tree.
- **Parameter**
  Pointer to "PEGeneralPrintWindowEventInfo" on page 467.
- **Return**
  Ignored.

PE_LAST_PAGE_BUTTON_CLICKED_EVENT

- **Called**
  After clicking the last page button; before going to the last page.
- **Parameter**
  Pointer to "PEGeneralPrintWindowEventInfo" on page 467.
- **Return**
  TRUE to use the default action; FALSE to cancel the default action.

PE_LAUNCH_SEAGATE_ANALYSIS_EVENT

- **Called**
  Whenever the Launch Seagate Analysis toolbar button is clicked.

  Note: Seagate Analysis is now known as Crystal Analysis.
- **Parameter**
  Pointer to"PELaunchSeagateAnalysisEventInfo" on page 479.
- **Return**
  TRUE to use the default action; FALSE to cancel the default action.

PE_LEFT_CLICK_EVENT

- **Called**
  Whenever the left mouse button has been clicked over the preview window.
- **Parameter**
  Pointer to"PEMouseClickEventInfo" on page 481.
- **Return**
  TRUE to use the default action; FALSE to cancel the default action.

PE_MAPPING_FIELD_EVENT

- **Called**
  Whenever the user calls PEVerifyDatabase and the field mapping method has been set to PE_FM_EVENT_DEFINED_FLD_MAP. The field mapping method can be retrieved and set with "PEGetFieldMappingType" on page 306, and "PESetFieldMappingType" on page 401.
- **Parameter**
  Pointer to"PEFieldMappingEventInfo" on page 462.
- **Return**
  TRUE to use the default action; FALSE to cancel the default action.

PE_MIDDLE_CLICK_EVENT

- **Called**
  Whenever the middle mouse button has been clicked over the preview window.
- **Parameter**
  Pointer to"PEMouseClickEventInfo" on page 481.
- **Return**
  TRUE to use the default action; FALSE to cancel the default action.

PE_NEXT_PAGE_BUTTON_CLICKED_EVENT

- **Called**
  After clicking the next page button, before going to the next page.
- **Parameter**
  Pointer to "PEGeneralPrintWindowEventInfo" on page 467.
- **Return**
  TRUE to use the default action; FALSE to cancel the default action.

PE_PREVIOUS_PAGE_BUTTON_CLICKED_EVENT

- **Called**
  After clicking the previous page button; before going to the previous page.
- **Parameter**
  Pointer to "PEGeneralPrintWindowEventInfo" on page 467.
- **Return**
  TRUE to use the default action; FALSE to cancel the default action.

PE_PRINT_BUTTON_CLICKED_EVENT

- **Called**
  After the Print button is clicked; before printing process starts.
- **Parameter**
  Pointer to "PEGeneralPrintWindowEventInfo" on page 467.
- **Return**
  TRUE to use the default action; FALSE to cancel the default action.

PE_PRINT_SETUP_BUTTON_CLICKED_EVENT

- **Called**
  After clicking Print Setup button; before showing the Print Setup dialog box.
- **Parameter**
  Pointer to “PEGeneralPrintWindowEventInfo” on page 467.
- **Return**
  TRUE to use the default action; FALSE to cancel the default action.

PE_READING_RECORDS_EVENT

- **Called**
  This event is fired during a reading database or regenerating saved data process. It is fired after a specified amount of time, not aver reading every record.
- **Parameter**
  Pointer to “PEReadingRecordsEventInfo” on page 491.
- **Return**
  Ignored.

PE_REFRESH_BUTTON_CLICKED_EVENT

- **Called**
  After clicking the Refresh button; before refreshing the data.
- **Parameter**
  Pointer to “PEGeneralPrintWindowEventInfo” on page 467.
- **Return**
  TRUE to use the default action; FALSE to cancel the default action.

PE_RIGHT_CLICK_EVENT

- **Called**
  Whenever the right mouse button has been clicked over the preview window.
- **Parameter**
  Pointer to“PEMouseClickEventInfo” on page 481.
- **Return**
  TRUE to use the default action; FALSE to cancel the default action.

PE_SEARCH_BUTTON_CLICKED_EVENT

- **Called**
  After the search button is clicked; before the search starts.
- **Parameter**
  Pointer to “PESearchButtonClickedEventInfo” on page 500.
- **Return**
  TRUE to use the default action; FALSE to cancel the default action.

PE_SHOW_GROUP_EVENT

- **Called**
  After clicking one of the group tree nodes; before showing that group.
- **Parameter**
  Pointer to "PEShowGroupEventInfo" on page 505.
- **Return**
  TRUE to use the default action. FALSE to cancel the default action.

PE_START_EVENT

- **Called**
  Before the Report Engine starts a process. A process can be printing to printer, exporting, printing to a window, or generating pages when navigating through the preview window.
- **Parameter**
  Pointer to"PEStartEventInfo" on page 506.
- **Return**
  TRUE to use the default action; FALSE to cancel the default action.

PE_STOP_EVENT

- **Called**
  Whenever a process has finished. Used in conjunction with PE_START_EVENT.
- **Parameter**
  Pointer to"PEStopEventInfo" on page 506.
- **Return**
  Ignored.

PE_ZOOM_LEVEL_CHANGING_EVENT

- **Called**
  After changing zoom control; before changing preview zoom level.
- **Parameter**
  Pointer to "PEZoomLevelChangingEventInfo" on page 521.
- **Return**
  Ignored.

Delphi Syntax

```
function PESetEventCallback(
    printJob: Word;
    callbackProc: pointer
        {Callback Function should be of form:
        Function callbacProc(eventID: smallint;
                    param: pointer;
                    userData: pointer)}
    ): Bool stdcall;
```

PESetFieldMappingType

Use PESetFieldMappingType to set the field mapping type code for the specified report.

C Syntax

```
BOOL CRPE_API PESetFieldMappingType
        short printJob,
        WORD mappingType );
```

Parameters

printJob	Specifies the print job for which you want to set the field mapping type code.
mappingType	The field mapping type code that you want to set. Use one of the PE_FM_XXX "Field Mapping Type Constants" on page 551.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

- Unmapped report fields will be removed.
- If mappingType = PE_FM_EVENT_DEFINED_FLD_MAP, you need to activate the PE_MAPPING_FIELD_EVENT and define a callback function.

VB Syntax

```
Declare Function PESetFieldMappingType Lib "crpe32.dll" (
    ByVal printJob As Integer, ByVal mappingType As Integer ) As Integer
```

Delphi Syntax

```
function PESetFieldMappingType (
    printJob: smallint;
    mappingType: Word
    ): BOOL stdcall;
```

PESetFont

Use PESetFont to set the font for field and / or text characters in the report section(s) specified. Use this call any time you need to change a default font at runtime in response to user input or to specify a built-in printer font.

C Syntax

```
BOOL CRPE_API PESetFont (
        short printJob,
        short sectionCode,
        short scopeCode,
        const char FAR *faceName,
        short fontFamily,
        short fontPitch,
        short charSet,
        short pointSize,
        short isItalic,
        short isUnderlined,
        short isStruckOut,
        short weight );
```

Parameters

printJob	Specifies the print job for which you want to select a font.	
sectionCode	Specifies the "Section Codes" on page 559 for the report section(s) for which you want to select a font. See "Working with section codes" on page 46.	
scopeCode	Specifies whether the font selected is to apply to fields only, to text only, or to both fields and text. To specify both fields and text, use the OR operator. Use one of the following codes.	
	Constant	**Description**
	PE_FIELDS	Sets the default font for all field values in the report section specified.
	PE_TEXT	Sets the default font for all text (that has not been entered as a field value) in the report section specified.
faceName	Specifies a pointer to the actual face name of the font you want to use. The face name you pass can typically come from a Font dialog box, be hard coded in the application or be chosen by the application from the fonts supported on the printer. Example: "Times New Roman". Pass 0 for no change.	
fontFamily	Specifies the font family for the font you want to use. Use one of the following FF_XXX constants.	
	Constant	**Description**
	FF_DONTCARE	No change.
	FF_ROMAN	Variable pitch font with serifs.
	FF_SWISS	Fixed pitch font without serifs.
	FF_MODERN	Fixed-pitch font, with or without serifs.
	FF_SCRIPT	Handwriting-like font.
	FF_DECORATIVE	Fancy display font.
fontPitch	Specifies the font pitch you wish to use. Use a constant value for the font pitch as defined in WINDOWS.H. Use DEFAULT_PITCH if you wish to retain the current default.	

	Constant	Description
	DEFAULT_PITCH	0X00
	FIXED_PITCH	0X01
	VARIABLE_PITCH	0X02
charSet	Specifies the character set you wish to use. Use a constant value for the character set as defined in WINDOWS.H. Use DEFAULT_CHARSET if you wish to retain the current default.	
	Constant	**Value**
	ANSI_CHARSET	0
	DEFAULT_ CHARSET	1
	SYMBOL_ CHARSET	2
	SHIFTJIS_ CHARSET	128
	HANGEUL_ CHARSET	129
	CHINESEBIG5_ CHARSET	136
	OEM_CHARSET	255
pointSize	Specifies the desired point size for the selected font. Pass 0 for no change.	
isItalic	Specifies whether the font selected should be italicized. Pass TRUE for Italic font, FALSE for non-Italic font, or PE_UNCHANGED to use the current default setting.	
isUnderlined	Specifies whether the font selected should be underlined. Pass TRUE for Underline, FALSE for no Underline, or PE_UNCHANGED to use the current default setting.	
isStruckOut	Specifies whether the font selected should be struck out. Pass TRUE for StrickOut, FALSE for no StrickOut, or PE_UNCHANGED to use the current default setting.	
weight	Specifies the weight of the font. Use a constant value from the weight values defined in WINDOWS.H. Pass 0 for no change.	
	Constant	**Value**
	FW_DONTCARE	0
	FW_THIN	100
	FW_EXTRALIGHT	200
	FW_LIGHT	300
	FW_NORMAL	400
	FW_MEDIUM	500
	FW_SEMIBOLD	600

FW_BOLD	700
FW_EXTRABOLD	800
FW_HEAVY	900
FW_ULTRALIGHT	FW_EXTRALIGHT
FW_REGULAR	FW_NORMAL
FW_DEMIBOLD	FW_SEMIBOLD
FW_ULTRABOLD	FW_EXTRABOLD
FW_BLACK	FW_HEAVY

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

This command includes a number of parameters:

- For the fontFamily, fontPitch, charSet, and weight parameters, use constant values from the font family, pitch, character set, and width defined in WINDOWS.H. Use 0 for each parameter that is not to be changed from the current default.
- For the faceName parameter, enter the actual name of the font. Enter 0 for no change.
- faceName, fontFamily, fontPitch, and charSet should all be specified whenever one of these parameters is specified. Use fontFamily = FF_DONTCARE, fontPitch = DEFAULT_PITCH, or charSet = DEFAULT_CHARSET to leave the default values unchanged.
- This function should be called before “PEStartPrintJob” on page 448, or the results may be inconsistent or unexpected.

VB Syntax

```
Declare Function PESetFont Lib “crpe32.dll” ( ByVal printJob As Integer,
    ByVal sectionCode As Integer, ByVal ScopeCode As Integer,
    ByVal FaceName As String, ByVal FontFamily As Integer,
    ByVal FontPitch As Integer, ByVal CharSet As Integer,
    ByVal PointSize As Integer, ByVal isItalic As Integer,
    ByVal isUnderlined As Integer, ByVal isStruckOut As Integer,
    ByVal Weight As Integer ) As Integer
```

Delphi Syntax

```
function PESetFont(
    printJob: Word;
    sectionCode: integer;
    scopeCode: integer;
    faceName: PChar;
    fontFamily: integer;
    fontPitch: integer;
    charSet: integer;
    pointSize: integer;
    isItalic: integer;
    isUnderlined: integer;
    isStruckOut: integer;
    weight: integer
    ): Bool stdcall;
```

dBASE for Windows Syntax

```
EXTERN CLOGICAL PESetFont (CWORD, CWORD, CWORD, CSTRING, CWORD, CWORD,
CWORD, CWORD, CWORD, CWORD, CWORD, CWORD) CRPE.DLL
```

PESetFormula

Use PESetFormula to change the specified formula to the formula string you supply as a parameter. This function will only change the text of a formula which already exists in the report; you cannot use it to add a new formula. This function can be used by itself to replace the formula string for a known formula.

This function can also be used as one of a series of functions (“PEGetFormula” on page 307; “PEGetHandleString” on page 318; and PESetFormula). The series can be used in a Custom-Print Link to identify and then change an existing formula at print time in response to a user selection. When you give the user the ability to change the formula at print time, your link must include code to replace formulaString with a user-generated value.

C Syntax

```
BOOL CRPE_API PESetFormula (
        short printJob,
        const char *formulaName,
        const char FAR *formulaString );
```

Parameters

printJob	Specifies the print job for which you want to set a new formula string.
formulaName	Specifies a pointer to the null-terminated string that contains the name of the formula for which you want to set a new formula string.
formulaString	Specifies a pointer to the null-terminated string that you want to replace the existing formula string.

Returns

- TRUE if the call is successful.
- FALSE if the named formula does not exist.
- Error code PE_ERR_BADFORMULANAME if the formula does not exist.
- Error code PE_ERR_BADFORMULATEXT if there is an error in the formula.

Remarks

- This function should be called before "PEStartPrintJob" on page 448 or the results may be inconsistent or unexpected.
- You cannot use this function to set conditional formulas.

VB Syntax

```
Declare Function PESetFormula Lib "crpe32.dll" (ByVal printJob As
Integer, ByVal FormulaName As String, ByVal FormulaString As String) As
Integer
```

Delphi Syntax

```
function PESetFormula (
    printJob: Word;
    formulaName: PChar;
    formulaString: PChar
    ): Bool stdcall;
```

dBASE for Windows Syntax

```
EXTERN CLOGICAL PESetFormula (CWORD, CSTRING, CSTRING) CRPE.DLL
```

PESetFormulaSyntax

Use PESetFormulaSyntax to set the formula syntax information to use for the next and all subsequent formula API call.

C Syntax

```
BOOL CRPE_API PESetFormulaSyntax (
    short printJob,
    PEFormulaSyntax FAR *formulaSyntax );
```

Parameters

printJob	Specifies the print job for which you want to set formula syntax.
formulaSyntax	Specifies a pointer to "PEFormulaSyntax" on page 466, which will contain the information that you want to set.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

- If any Formula API is called before PESetFormulaSyntax is called, then the default PE_FST_CRYSTAL is assumed.
- For running total condition formula:
  - formulaSyntax[0] is the syntax for the evalFormula.
  - formulaSyntax[1] is the syntax for the reset formula.

VB Syntax

```
Declare Function PESetFormulaSyntax Lib "crpe32.dll" (
    ByVal printJob As Integer, formulaSyntax As PEFormulaSyntax ) As
Integer
```

PESetGraphAxisInfo

Use PESetGraphAxisInfo to set the several chart axis options that are available.

C Syntax

```
BOOL CRPE_API PESetGraphAxisInfo (
        short printJob,
        short sectionN,
        short graphN,
        PEGraphAxisInfo FAR * graphAxisInfo );
```

Parameters

printJob	Specifies the print job for which you want to set chart axis information.
sectionN	Specifies the section of the report containing the chart for which you want to set chart axis information.
graphN	Specifies for which chart within the section you want to set the chart axis information. This value is 0-based. Charts are numbered based on their order of insertion into the report.
graphAxisInfo	Specifies a pointer to “PEGraphAxisInfo” on page 468, which will contain the new information.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PESetGraphAxisInfo Lib "crpe32.dll" (
    ByVal printJob As Integer, ByVal sectionN As Integer,
    ByVal graphN As Integer, graphAxisInfo As PEGraphAxisInfo ) As
Integer
```

Delphi Syntax

```
function PESetGraphAxisInfo (
  printJob          : Smallint;
  sectionN          : Smallint;
  graphN            : Smallint;
  var graphAxisInfo : PEGraphAxisInfo): Bool; {$ifdef WIN32} stdcall;
{$endif}
```

PESetGraphFontInfo

Use PESetGraphFontInfo to set the font information for the specified chart.

C Syntax

```
BOOL CRPE_API PESetGraphFontInfo (
        short printJob,
        short sectionN,
        short graphN,
        WORD titleFontType,
        PEFontColorInfo FAR *fontColourInfo );
```

Parameters

printJob	Specifies the print job for which you want to set chart font information.
sectionN	Specifies the number of the section in which the chart appears. This parameter should be within the range obtained by “PEGetNSections” on page 328.
graphN	Specifies for which chart within the section you want to set the font information. This value is 0-based. Charts are numbered based on their order of insertion into the report.
titleFontType	Uses one of the PE_GTF_XXX “Graph Text Font Constants” on page 554
fontColourInfo	Specifies a pointer to PEFontColorInfo, which will contain the new information.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PESetGraphFontInfo Lib "crpe32.dll" (
    ByVal printJob As Integer, ByVal sectionN As Integer,
    ByVal graphN As Integer, ByVal titleFontType As Integer,
    fontColourInfo As PEFontColorInfo ) As Integer
```

Delphi Syntax

```
function PESetGraphFontInfo (
  printJob           : Smallint;
  sectionN           : Smallint;
  graphN             : Smallint;
  titleFontType      : Word;
  var fontColourInfo : PEFontColorInfo): Bool; {$ifdef WIN32}
stdcall;{$endif}
```

PESetGraphOptionInfo

Use PESetGraphOptionInfo to set display options for the specified chart.

C Syntax

```
BOOL CRPE_API PESetGraphOptionInfo (
        short printJob,
        short sectionN,
        short graphN,
        PEGraphOptionInfo FAR *graphOptionInfo );
```

Parameters

printJob	Specifies the print job for which you want to set chart display information.
sectionN	Specifies the number of the section in which the chart appears. This parameter should be within the range obtained by PEGetNSections.
graphN	Specifies for which chart within the section you want to set the chart display information. This value is 0-based. Charts are numbered based on their order of insertion into the report.
graphOptionInfo	Specifies a pointer to “PEGraphOptionInfo” on page 471, which will contain the new information.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PESetGraphOptionInfo Lib "crpe32.dll" (
    ByVal printJob As Integer, ByVal sectionN As Integer,
    ByVal graphN As Integer, graphOptionInfo As PEGraphOptionInfo ) As
Integer
```

Delphi Syntax

```
function PESetGraphOptionInfo (
  printJob           : Smallint;
  sectionN           : Smallint;
  graphN             : Smallint;
  var graphOptionInfo : PEGraphOptionInfo): Bool; {$ifdef WIN32}
         stdcall; {$endif}
```

PESetGraphTextDefaultOption

Use PESetGraphTextDefaultInfo to enable or disable chart title defaults.

C Syntax

```
BOOL CRPE_API PESetGraphTextDefaultOption (
    short printJob,
    short sectionN,
    short graphN,
    WORD titleType,
    BOOL useDefault );
```

Parameters

printJob	Specifies the print job for which you want to enable/disable chart title default option information.
sectionN	Specifies the 0-based index number of the section in which the chart appears. This parameter should be within the range obtained by "PEGetNSections" on page 328.
graphN	Specifies the 0-based index number of the chart for which you want to enable/disable the chart text default option information. Charts are numbered based on their order of insertion into the report.
titleType	Specifies the title type. Use one of the PE_GTT_XXX "Graph Title Type Constants" on page 555.
useDefault	Specifies the Boolean value indicating whether or not chart title defaults are enabled.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PESetGraphTextDefaultOption Lib "crpe32.dll" (
    ByVal printJob As Integer, ByVal sectionN As Integer,
    ByVal graphN As Integer, ByVal titleType As Integer,
    ByVal useDefault As Long ) As Integer
```

PESetGraphTextInfo

Use PESetGraphTextInfo to set the title text information for the specified chart.

C Syntax

```
BOOL CRPE_API PESetGraphTextInfo (
        short printJob,
        short sectionN,
        short graphN,
        WORD titleType,
        LPCSTR title );
```

Parameters

printJob	Specifies the print job for which you want to set title text information.
sectionN	Specifies the 0-based index number of the section in which the chart appears. This parameter should be within the range obtained by "PEGetNSections" on page 328.
graphN	Specifies the 0-based index number of the chart for which you want to set the title text information. Charts are numbered based on their order of insertion into the report.
titleType	Specifies the title type. Use one of the PE_GTT_XXX "Graph Title Type Constants" on page 555.
title	Specifies a pointer to the string containing the title text.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PESetGraphTextInfo Lib "crpe32.dll" (
    ByVal printJob As Integer, ByVal sectionN As Integer,
    ByVal graphN As Integer, ByVal titleType As Integer,
    ByVal title As String ) As Integer
```

Delphi Syntax

```
function PESetGraphTextInfo (
  printJob  : Smallint;
  sectionN  : Smallint;
  graphN    : Smallint;
  titleType : Word;
  title     : PChar): Bool; {$ifdef WIN32} stdcall; {$endif}
```

PESetGraphTypeInfo

Use PESetGraphTypeInfo to set the type of the specified chart.

C Syntax

```
BOOL CRPE_API PESetGraphTypeInfo (
        short printJob,
        short sectionN,
        short graphN,
        PEGraphTypeInfo FAR *graphTypeInfo );
```

Parameters

printJob	Specifies the print job for which you want to set chart type information.
sectionN	Specifies the number of the section in which the chart appears. This parameter should be within the range obtained by "PEGetNSections" on page 328.
graphN	Specifies for which chart within the section you want to set the type. This value is 0-based. Charts are numbered based on their order of insertion into the report.
graphTypeInfo	Specifies a pointer to "PEGraphTypeInfo" on page 473, which will contain the new information.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PESetGraphTypeInfo Lib "crpe32.dll" (
    ByVal printJob As Integer, ByVal sectionN As Integer,
    ByVal graphN As Integer, graphTypeInfo As PEGraphTypeInfo ) As
Integer
```

Delphi Syntax

```
function PESetGraphTypeInfo (
  printJob          : Smallint;
  sectionN          : Smallint;
  graphN            : Smallint;
  var graphTypeInfo : PEGraphTypeInfo): Bool; {$ifdef WIN32} stdcall;
{$endif}
```

PESetGroupCondition

Use PESetGroupCondition to change the group condition for a group section. Use this function whenever you want to change the grouping at print time, for example, to print one report grouped in several different ways.

C Syntax

```
BOOL CRPE_API PESetGroupCondition (
        short printJob,
        short sectionCode,
        const char FAR *conditionField,
        short condition,
        short sortDirection );
```

Parameters

printJob	Specifies the print job for which you want to change the group condition for a group section.
sectionCode	Specifies the code for the report section for which you want to set the group condition. See "Working with section codes" on page 46.
conditionField	Specifies a pointer to the name of the field that triggers a summary whenever its value changes. This parameter is a result of calling "PEGetHandleString" on page 318, with conditionFieldHandle and conditionFieldLength, returned by "PEGetGroupCondition" on page 315.
condition	Specifies the condition that will trigger a summary. Use one of the PE_GC_XXX "Group Condition Constants" on page 556. Note that the constants available are different for different field types.
sortDirection	Use one of the PE_SF_XXX "Sort Order Constants" on page 560.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

- No default values are allowed. You must specify a value for all parameters when using this function.
- If you have a formula that references a summary field and you change the condition on the summary field without fixing the formula, you will get an error.
- This function should be called before "PEStartPrintJob" on page 448 or the results may be inconsistent or unexpected.

VB Syntax

```
Declare Function PESetGroupCondition Lib "crpe32.dll" (ByVal printJob As
Integer, ByVal sectionCode As Integer, ByVal ConditionField As String,
ByVal Condition As Integer, ByVal SortDirection As Integer) As Integer
```

Delphi Syntax

```
function PESetGroupCondition (
    printJob: Word;
    sectionCode: smallint;
    conditionField: PChar;
    condition: smallint;
    sortDirection: smallint
    ): Bool stdcall;
```

dBASE for Windows Syntax

```
EXTERN CLOGICAL PESetGroupCondition (CWORD, CWORD, CSTRING, CWORD, CWORD)
CRPE.DLL
```

PESetGroupOptions

Use PESetGroupOptions to set grouping options for the specified group.

C Syntax

```
BOOL CRPE_API PESetGroupOptions (
        short printJob,
        short groupN,
        PEGroupOptions FAR *groupOptions );
```

Parameters

printJob	Specifies the print job for which you wish to set grouping options.
groupN	Specifies the 0-based group level number.
groupOptions	Specifies a pointer to "PEGroupOptions" on page 474.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

- This function should be called before "PEStartPrintJob" on page 448 or the results may be inconsistent or unexpected.
- If you are using PESetGroupOptions to set the top/bottom N sort field, all the group sort fields related to the group will be deleted and a new one specified by the group options will be added.

VB Syntax

```
Declare Function PESetGroupOptions Lib "crpe32.dll" (ByVal printJob As
Integer, ByVal groupN As Integer, groupOptions As PEGroupOptions) As
Integer
```

Delphi Syntax

```
function PESetGroupOptions (
    printJob: Word;
    groupN: smallint;
    var groupOptions: PEGroupOptions
    ): Bool; stdcall;
```

PESetGroupSelectionFormula

Use PESetGroupSelectionFormula to change the group selection formula to the formula string you supply as a parameter. This function can be used by itself to replace an existing group selection formula and also as one of a series of functions ("PEGetGroupSelectionFormula" on page 317; "PEGetHandleString" on page 318; and PESetGroupSelectionFormula). The series can be used in a Custom-Print Link to identify and then change an existing group selection formula at print time in response to a user selection. When you give the user the ability to change the group selection formula at print time, your link must include code to replace formulaString with a user-generated value.

C Syntax

```
BOOL CRPE_API PESetGroupSelectionFormula (
        short printJob,
        const char FAR *formulaString );
```

Parameters

printJob	Specifies the print job for which you want to set a new group selection formula.
formulaString	Specifies a pointer to the null-terminated string you want to assign to the group selection formula.

Returns

- TRUE if the call is successful.
- FALSE if the call fails due to an internal error (for example, the connection to the database fails).

Remarks

- Immediately after setting the new formula with PESetGroupSelectionFormula, the new formula should be verified with "PECheckGroupSelectionFormula" on page 283.
- This function should be called before "PEStartPrintJob" on page 448 or the results may be inconsistent or unexpected.

VB Syntax

```
Declare Function PESetGroupSelectionFormula Lib "crpe32.dll" (ByVal printJob
As Integer, ByVal formulaString As String) As Integer
```

Delphi Syntax

```
function PESetGroupSelectionFormula (
    printJob: Word;
    formulaString: PChar
    ): Bool stdcall;
```

dBASE for Windows Syntax

```
EXTERN CLOGICAL PESetGroupSelectionFormula (CWORD, CSTRING) CRPE.DLL
```

PESetMargins

Use PESetMargins to set the page margins for the specified report to the values you supply as parameters. Use this function any time you want to set the printer margins at runtime in response to user specifications.

C Syntax

```
BOOL CRPE_API PESetMargins (
        short printJob,
        short left,
        short right,
        short top,
        short bottom );
```

Parameters

For each margin parameter, specify the margin in twips or PM_SM_DEFAULT to use the corresponding default margin for the currently selected printer. See Remarks below.

printJob	Specifies the print job for which you want to set new margins.
left	Specifies the left margin.
right	Specifies the right margin.
top	Specifies the top margin.
bottom	Specifies the bottom margin.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

- This function should be called before “PEStartPrintJob” on page 448, or the results may be inconsistent or unexpected.
- A twip is 1/1440 of an inch; there are 20 twips in a point. To set .5" margins, for example, you would enter the value 720.

VB Syntax

```
Declare Function PESetMargins Lib “crpe32.dll” ( ByVal printJob As
Integer,
    ByVal LeftMargin As Integer, ByVal RightMargin As Integer,
    ByVal TopMargin As Integer, ByVal BottomMargin As Integer ) As
Integer
```

Delphi Syntax

```
function PESetMargins (
    printJob: Word;
    left: Word;
    right: Word;
    top: Word;
    bottom: Word
    ): Bool stdcall;
```

dBASE for Windows Syntax

```
EXTERN CLOGICAL PESetMargins (CWORD, CWORD, CWORD, CWORD, CWORD)
CRPE.DLL
```

PESetNDetailCopies

Use PESetNDetailCopies to print multiple copies of the Details section of the report. For example, you can use this function to print multiple copies of labels for a customer, multiple copies of a purchase order, or multiple copies of anything set up in the Details section of your report. To retrieve the number of times each Details section is to be printed, use “PEGetNDetailCopies” on page 321.

C Syntax

```
BOOL CRPE_API PESetNDetailCopies (
        short printJob,
        short nDetailCopies );
```

Parameters

printJob	Specifies the print job for which you want set the number of copies to print.
nDetailCopies	Specifies the number of report copies you want to print.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

- This function should be called before "PEStartPrintJob" on page 448, or the results may be inconsistent or unexpected.
- To change top/bottom N group sorting order, use "PESetGroupOptions" on page 414.

VB Syntax

```
Declare Function PESetNDetailCopies Lib "crpe32.dll" (
    ByVal printJob As Integer, ByVal nDetailCopies As Integer ) As
Integer
```

Delphi Syntax

```
function PESetNDetailCopies(
    printJob: Word;
    nDetailCopies: smallint;
    ): Bool stdcall;
```

dBASE for Windows Syntax

```
EXTERN CLOGICAL PESetNDetailCopies (CWORD, CWORD) CRPE.DLL
```

PESetNthAlertConditionFormula

Use PESetNthAlertConditionFormula to set the condition formula for the specified Report Alert associated with the print job.

C Syntax

```
BOOL CRPE_API PESetNthAlertConditionFormula (short printJob,
                                             short alertN,
                                             constant char FAR * formula);
```

Parameters

printJob	Specifies the print job for which you want to set the Report Alert condition formula.
alertN	Specifies the Report Alert for which you want to set the Report Alert condition formula.
formula	Specifies a pointer to the string containing the condition formula to set for the Report Alert.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

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.

VB Syntax

```
Declare Function PESetNthAlertConditionFormula Lib "crpe32.dll" (ByVal
printJob%, ByVal alertN%, formula As String) As Integer
```

Delphi Syntax

```
function PESetNthAlertConditionFormula(
   printJob : Smallint;
   alertN : Smallint;
   const formula : PCHAR) : boolean stdcall;
```

PESetNthAlertDefaultMessage

Use PESetNthAlertDefaultMessage to set the default message for the specified Report Alert associated with the print job.

C Syntax

```
BOOL CRPE_API PESetNthDefaultMessage (short printJob,
                                      short alertN,
                                      constant char FAR * text);
```

Parameters

printJob	Specifies the print job for which you want to set the Report Alerts default message.
alertN	Specifies the Report Alert for which you want to set the default message.
text	Specifies a pointer to the string containing the default message for the Report Alert.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PESetNthAlertDefaultMessage Lib "crpe32.dll" (ByVal
printJob%, ByVal alertN%, text As String) As Integer
```

Delphi Syntax

```
function PESetNthAlertDefaultMessage(
   printJob : Smallint;
   alertN : Smallint;
   const text : PCHAR) : boolean stdcall;
```

PESetNthAlertMessageFormula

Use PESetNthAlertMessageFormula to set the message formula for the specified Report Alert associated with the print job.

C Syntax

```
BOOL CRPE_API PESetNthAlertMessageFormula (short printJob,
                                           short alertN,
                                           const char FAR *formula);
```

Parameters

printJob	Specifies the print job for which you want to set the Report Alert's message.formula.
alertN	Specifies the Report Alert for which you want to set the message.formula.
formula	Specifies a pointer to the string containing the message.formula for the Report Alert.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

The result of the formula must be a string. If PESetNthAlertMessageFormula is set it will override the value set for the Report Alerts default message.

VB Syntax

```
Declare Function PESetNthAlertMessageFormula Lib "crpe32.dll" (ByVal
printJob%, ByVal alertN%, formula As String) As Integer
```

Delphi Syntax

```
function PESetNthAlertMessageFormula(
   printJob : Smallint;
   alertN : Smallint;
   const formula : PCHAR) : boolean stdcall;
```

PESetNthGroupSortField

Use PESetNthGroupSortField to set one of the group sort fields in the specified report. This function can only be used to modify an existing sort field and direction when the sort field number, name, and direction are known.

Note: Note: PESetNthGroupSortField cannot be used to create a new sort field.

The function can also be used as one of a series of functions ("PEGetNGroupSortFields" on page 323, called once; "PEGetNthGroupSortField" on page 335, or "PEGetHandleString" on page 318, called as many times as needed to identify the correct group sort field; and PESetNthGroupSortField called once,

when the correct group sort field is identified). The series can be used in a Custom-Print Link to identify and then change an existing group sort field and/or sort order in response to a user selection at print time. When you give the user the ability to specify group sort field(s) and/or direction at print time, your link must include code to replace name and/or direction with user-generated values.

C Syntax

```
BOOL CRPE_API PESetNthGroupSortField (
        short printJob,
        short sortFieldN,
        const char FAR *name,
        short direction );
```

Parameters

printJob	Specifies the print job for which you want to set a group sort field.
sortFieldN	Specifies the 0-based number of the sort field you want to set. The first sort field is field 0. If the report has N sort fields, the function can be called with sortFieldN between 0 and N-1. If the report has N sort fields, you can call the function with sortFieldN = N to add a new sort field to the end of the list of existing sort fields. If N=0, the function will create the first sort field.
name	Specifies a pointer to the null-terminated string that contains the name of the group sort field.
direction	Specifies the sort directions. Use one of the PE_SF_XXX "Sort Order Constants" on page 560.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

This function should be called before "PEStartPrintJob" on page 448, or the results may be inconsistent or unexpected.

VB Syntax

```
Declare Function PESetNthGroupSortField Lib "crpe32.dll" ( _
    ByVal printJob As Integer, ByVal sortFieldN As Integer, _
    ByVal SortGroupName As String, ByVal Direction As Integer ) As
Integer
```

Delphi Syntax

```
function PESetNthGroupSortField (
    printJob: Word;
    sortFieldN: smallint;
     name: PChar;
     direction: smallint
     ): Bool stdcall;
```

dBASE for Windows Syntax

```
EXTERN CLOGICAL PESetNthGroupSortField (CWORD, CWORD, CSTRING, CWORD)
CRPE.DLL
```

PESetNthParameterDefaultValue

Use PESetNthParameterDefaultValue to set a default value for a specified parameter field in a report. Use "PEGetNParameterDefaultValues" on page 326, to retrieve the number of default values for the parameter field.

C Syntax

```
BOOL CRPE_API PESetNthParameterDefaultValue (
        short printJob,
        const char FAR *parameterFieldName,
        const char FAR *reportName,
        short index,
        PEValueInfo FAR *valueInfo );
```

Parameters

printJob	Specifies the print job for which you want to set a parameter default value.
parameterField Name	Specifies a pointer to the string containing the parameter field name.
reportName	Specifies a pointer to the string containing the report name. See Remarks below.
index	Specifies the index number of the default value to be set.
valueInfo	Specifies a pointer to "PEValueInfo" on page 516, which will contain the default value.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

Regarding parameter reportName:

- For the main report, pass an empty string ("").
- For a subreport, pass the file path and name of the subreport as a NULL-terminated string.

VB Syntax

```
Declare function PESetNthParameterDefaultValue Lib "crpe32.dll" ( _
    ByVal printJob As Integer, ByVal parameterFieldName As String, _
    ByVal reportName As String, ByVal index As Integer, _
    valueInfo As PEValueInfo ) As Integer
```

Delphi Syntax

```
function PESetNthParameterDefaultValue (
    printJob: smallint;
    const parameterFieldName: PChar;
    const reportName: PChar;
    index: smallint;
    var valueInfo: PEValueInfo
    ): BOOL stdcall;
```

PESetNthParameterField

Use PESetNthParameterField to set a value for the specified parameter field. For new development, see Remarks below.

C Syntax

```
BOOL CRPE_API PESetNthParameterField (
        short printJob,
        short parameterN,
        PEParameterFieldInfo FAR *parameterInfo );
```

Parameters

printJob	Specifies the print job for which you want to set a parameter field value.
parameterN	Specifies the number of the parameter field in the report.
parameterInfo	Specifies a pointer to “PEParameterFieldInfo” on page 484 which is used to pass the parameter field value information. See Remarks below.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

- For new development, the Default/CurrentValueSet members of PEParameterFieldInfo must be set to FALSE and the following new calls used to access default and current value lists.
  - “PEAddParameterCurrentRange” on page 278
  - “PEAddParameterCurrentValue” on page 279
  - “PEAddParameterDefaultValue” on page 280
  - “PESetNthParameterDefaultValue” on page 422
- To if you wish to set the parameter field to NULL then use CRWNULL (for example, ParameterFieldInfo.currentValue = CRWNULL). It is of Type String and independent of the data type of the parameter. See “PEParameterFieldInfo” on page 484.

- To determine if a parameter field is a stored procedure, use "PEGetNthParameterType" on page 341 or "PEGetNthParameterField" on page 340 functions.
- This function should be called before "PEStartPrintJob" on page 448 or the results may be inconsistent or unexpected.

Visual Basic Syntax

```
Declare Function PESetNthParameterField Lib "crpe32.dll" _
    (ByVal printJob As Integer, ByVal varN As Integer, varInfo As _
    PEParameterFieldInfo) As Integer
```

Delphi Syntax

```
function PESetNthParameterField (
    printJob: Word;
    varN: Smallint;
    var varInfo: PEParameterFieldInfo
    ): Bool stdcall;
```

PESetNthParameterValueDescription

Use PESetNthParameterValueDescription to set the description of the value for a parameter.

C Syntax

```
BOOL CRPE_API PESetNthParameterValueDescription (
        short printJob,
        const char FAR *parameterFieldName,
        const char FAR *reportName,
        short index,
        char FAR *valueDesc );
```

Parameters

printJob	Specifies the print job for which you want to set parameter value description information.
parameterFieldName	Specifies a pointer to the parameterFieldName for which you want to set the parameter value description.
reportName	Specifies a pointer to the report name. See Remarks below.
index	Specifies the index.
valueDesc	Specifies a pointer to the handle of the value description to be set.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

Regarding parameter reportName:

- For the main report, pass an empty string ("").
- For a subreport, pass the file path and name of the subreport as a NULL-terminated string.

VB Syntax

```
Declare Function PESetNthParameterValueDescription Lib "crpe32.dll" (_
    ByVal printJob As Integer, ByVal parameterFieldName As String, _
    ByVal reportName As String, ByVal index As Integer, _
    ByVal valueDesc As String ) As Integer
```

Delphi Syntax

```
function PESetNthParameterValueDescription (
  printJob            : Smallint;
  parameterFieldName  : PChar;
  reportName          : PChar;
  index               : Smallint;
  valueDesc           : PChar): Bool; {$ifdef WIN32} stdcall; {$endif}
```

PESetNthSortField

Use PESetNthSortField to set one of the sort fields in the specified report. This function can be used by itself to set a sort field / direction when there is not one already set, or to change a sort field / direction when the number and name of the sort field are known.

The function can also be used as one of a series of functions (“PEGetNSortFields” on page 330, called once; “var reportAlertInfo : PEReportAlertInfo) : boolean stdcall;” on page 343, or “PEGetHandleString” on page 318, called together as many times as needed to identify the correct sort field; and PESetNthSortField, called once when the correct sort field is identified). The series can be used in a Custom-Print Link to identify and then change an existing sort field and / or sort order in response to a user selection at print time. When you give the user the ability to specify sort field(s) and / or direction at print time, your link must include code to replace name and / or sort direction with user-generated values.

C Syntax

```
BOOL CRPE_API PESetNthSortField (
        short printJob,
        short sortFieldN,
        const char FAR *name,
        short direction );
```

Parameters

printJob	Specifies the print job for which you want to set sort field information.
sortFieldN	Specifies the 0-based number of the sort fields that you want to set. The first sort field is field 0. If the report has N sort fields, the function can be called with sortFieldN between 0 and N-1 to replace an existing sort field. If the report has N sort fields, you can call the function with sortFieldN = N to add a new sort field to the end of the list of existing sort fields. If N=0, the function will add the first sort field.
name	Specifies a pointer to the null-terminated string that contains the name of the sort field.
direction	Specifies the sort direction. Use one of the PE_SF_XXX "Sort Order Constants" on page 560.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

This function should be called before "PEStartPrintJob" on page 448 or the results may be inconsistent or unexpected.

VB Syntax

```
Declare Function PESetNthSortField Lib "crpe32.dll" ( _
    ByVal printJob As Integer, ByVal sortNumber As Integer, _
    ByVal SortFieldName As String, ByVal Direction As Integer ) As
Integer
```

Delphi Syntax

```
function PESetNthSortField (
    printJob: Word;
    sortFieldN: smallint;
    name: PChar;
    direction: smallint
    ): Bool stdcall;
```

dBASE for Windows Syntax

```
EXTERN CLOGICAL PESetNthSortField (CWORD, CWORD, CSTRING, CWORD) CRPE.DLL
```

PESetNthTableLocation

Use PESetNthTableLocation to set the location for a selected table in the specified print job. This function is typically combined with "PEGetNthTableLocation" on page 347 to identify the location of a table and then to change it.

C Syntax

```
BOOL CRPE_API PESetNthTableLocation (
        short printJob,
        short tableN,
        PETableLocation FAR *location );
```

Parameters

printJob	Specifies the handle of the print job for which you want to set a table's location.
tableN	Specifies the 0-based number of the table for which you want to set a new location. The first table is table 0. The last table is N-1.
location	Specifies the pointer to "PETableLocation" on page 510. The format of the string will be depend on the type of database specified.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PESetNthTableLocation Lib "crpe32.dll" ( ByVal printJob
As Integer, ByVal TableN As Integer, Location As PETableLocation ) As
Integer
```

Delphi Syntax

```
function PESetNthTableLocation(
    printJob: Word;
    tableN: smallint;
     var location: PETableLocation
     ): Bool stdcall;
```

PESetNthTableLogOnInfo

Use PESetNthTableLogOnInfo to set the log on information for the specified print job to the values in "PELogOnInfo" on page 479.

C Syntax

```
BOOL CRPE_API PESetNthTableLogOnInfo (
    short printJob,
    short tableN,
    PELogOnInfo FAR *logOnInfo,
    BOOL propagateAcrossTables );
```

Parameters

printJob	Specifies the print job for which you want to set table log on information.
tableN	Specifies the 0-based number of the table for which you want to set log on information. The first table is table 0. The last table is N-1.
logOnInfo	Specifies a pointer to the "PELogOnInfo" on page 479.
propagateAcrossTables	If set to TRUE, the program will apply the new log on information to any other tables in the report that had the same original server and database names as the specified table. If set to FALSE, the program will apply the new log on information only to the table specified.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

- See "PELogOnServer" on page 374 for additional comments regarding "PELogOnInfo" on page 479.
- The program logs on when printing the report, but you must first set the correct log on information using PESetNthTableLogOnInfo. Logging off is performed automatically when the print job is closed.
- You must supply at least the password with this function. You can pass empty strings ("") for the other parameters or, alternatively, you can change the server, database, and/or user ID by entering the appropriate strings.
- When you create a report from a single database (for example, one .MDB file with multiple tables), set the propagateAcrossTables parameter to TRUE. This insures that the changes are made to all tables in the .MDB file (thus avoiding the necessity to code the changes for each table individually).
- This function can be used to set the location of an Essbase application and database used by a report. For complete information, see "PELogOnInfo" on page 479.

VB Syntax

```
Declare Function PESetNthTableLogOnInfo Lib "crpe32.dll" ( ByVal
printJob As Integer, ByVal TableN As Integer, LogOnInfo As PELogOnInfo,
ByVal Propagate As Integer ) As Integer
```

Delphi Syntax

```
function PESetNthTableLogOnInfo (
    printJob: Word;
    tableN: smallint;
    var logOnInfo: PELogOnInfo;
    propagateAcrossTables: Bool
    ): Bool stdcall;
```

PESetNthTablePrivateInfo

Use PESetNthTablePrivateInfo to set the information needed for using data objects such as ADO, RDO, or CDO with the Active Data Driver (PS2MON.DLL).

C Syntax

```
BOOL CRPE_API PESetNthTablePrivateInfo (
        short printJob,
        short tableN,
        PETablePrivateInfo FAR *privateInfo );
```

Parameters

printJob	Specifies the print job for which you want to change the MS Access session information.
tableN	Specifies the 0-based number of the table for which you want to set table private information. The first table is table 0. The last table is N-1.
privateInfo	Specifies a pointer to “PETablePrivateInfo” on page 511.

Returns

- TRUE if the call is successful.
- FALSE if the call fails

Remarks

The PESetNthTablePrivateInfo function is used in conjunction with the Crystal Report Print Engine (Crpe32.dll) in a Visual C++ application. If you are using Visual Basic see “SetActiveDataSource” on page 604.

Delphi Syntax

```
function PESetNthTablePrivateInfo (
    printJob: Smallint;
    tableN:   Smallint;
    var privateInfo:          PETablePrivateInfo
)    :Bool; {$ifdef WIN32} stdcall; {$endif}
```

PESetNthTableSessionInfo

Use PESetNthTableSessionInfo to set the specified session information when opening a Microsoft Access table. Many Microsoft Access database tables require that a session be opened before the table can be used. Use PESetNthTableSessionInfo to open the session when required.

C Syntax

```
BOOL CRPE_API PESetNthTableSessionInfo (
        short printJob,
        short tableN,
        PESessionInfo FAR *sessionInfo,
        BOOL propagateAcrossTables );
```

Parameters

printJob	Specifies the print job for which you want to change the MS Access session information.
tableN	Specifies the 0-based number of the table for which you want to open the session. The first table is table 0. The last table is N-1.
sessionInfo	Specifies a pointer to "PESessionInfo" on page 503.
propagateAcrossTables	Boolean value indicating whether the session information should be used for opening all tables being used in the report.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

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 Password member before calling "PESetNthTableSessionInfo" on page 429.
- If the Access database contains database-level security, use a newline character, '\n' (ASCII character 10) followed by the database-level password (for example, "\ndbpassword").
  - If the Access database contains both session security and database-level security, use the session password followed by the newline character and the database password (for example, "sesspswd\ndbpassword").
- Alternately, database-level security can also be handled by assigning the database-level password to the Password member of "PELogOnInfo" on page 479 and calling "PESetNthTableLogOnInfo" on page 427.

VB Syntax

```
Declare Function PESetNthTableSessionInfo Lib "crpe32.dll" ( _
    ByVal printJob As Integer, ByVal TableN As Integer, _
    SessionInfo As PESessionInfo, ByVal PropagateAcrossTables _
    As Integer ) As Integer
```

Delphi Syntax

```
function PESetNthTableSessionInfo (
    printJob: Word;
    tableN: smallint;
    var sessionInfo: PESessionInfo;
    propagateAcrossTables: Bool
    ): Bool stdcall;
```

PESetParameterMinMaxValue

Use PESetParameterMinMaxValue to set the minimum and / or maximum possible values for the specified parameter in a report.

C Syntax

```
BOOL CRPE_API PESetParameterMinMaxValue(
        short printJob,
        const char FAR *parameterFieldName,
        const char FAR *reportName,
        PEValueInfo FAR *valueMin,
        PEValueInfo FAR *valueMax );
```

Parameters

printJob	Specifies the print job for which you want to set minimum and / or maximum parameters values.
parameterFieldName	Specifies a pointer to the string containing the parameter name.
reportName	Specifies a pointer to the string containing the report name. See Remarks below.
valueMin	Specifies a pointer to “PEValueInfo” on page 516, which contains minimum value information. See Remarks below.
valueMax	Specifies a pointer to “PEValueInfo” on page 516, which contains maximum value information. See Remarks below.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

- Regarding parameter reportName:
  - For the main report, pass an empty string ("").
  - For a subreport, pass the file path and name of the subreport as a NULL-terminated string.
- Regarding parameters valueMin and valueMax:
  - Set valueMin to NULL if specifying maximum value only; valueMin must be non NULL if valueMax is NULL.
  - Set valueMax to NULL if specifying minimum value only; valueMax must be non-NULL if valueMin is NULL.
  - If both valueMin and valueMax are specified (that is, non-NULL), then the valueType field of both structures must be the same or error code PE_ERR_INCONSISTANTTYPES is returned.

VB Syntax

```
Declare Function PESetParameterMinMaxValue Lib "crpe32.dll" ( _
    ByVal printJob As Integer, ByVal parameterFieldName As String, _
    ByVal reportName As String, valueMin As PEValueInfo, _
    valueMax As PEValueInfo ) As Integer
```

Delphi Syntax

```
function PESetParameterMinMaxValue (
    printJob: smallint;
    const parameterFieldName: PChar;
    const reportName: PChar;
    var valueMin: PEValueInfo;
    var valueMax: PEValueInfo): BOOL stdcall;
```

PESetParameterPickListOption

Use PESetParameterPickListOption to set the parameter pick list options for a report. This function sets the values in "PEParameterPickListOption" on page 487.

C Syntax

```
BOOL CRPE_API PESetParameterPickListOption (
        short printJob,
        const char FAR *parameterFieldName,
        const char FAR *reportName,
        PEParameterPickListOption FAR *pickListOption );
```

Parameters

printJob	Specifies the print job for which you want to set the parameter pick list options.
parameterFieldName	Specifies a pointer to the parameterFieldName for which you want to set pick list options.
reportName	Specifies a pointer to the report name. See Remarks below.
pickListOption	Specifies a pointer to "PEParameterPickListOption" on page 487, which will contain the new information.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

Regarding parameter reportName:

- For the main report, pass an empty string ("").
- For a subreport, pass the file path and name of the subreport as a NULL-terminated string.

VB Syntax

```
Declare Function PESetParameterPickListOption Lib "crpe32.dll" ( _
    ByVal printJob As Integer, ByVal parameterFieldName As String, _
    ByVal reportName As String, pickListOption As
PEParameterPickListOption _
    ) As Integer
```

Delphi Syntax

```
function PESetParameterPickListOption (
  printJob           : Smallint;
  parameterFieldName : PChar;
  reportName         : PChar;
  var pickListOption : PEParameterPickListOption): Bool; {$ifdef WIN32}
        stdcall; {$endif}
```

PESetParameterValueInfo

Use PESetParameterValueInfo to set information about the values which can be stored in a specified parameter field. For example, it establishes whether fields are editable, nullable, or can have multiple values, etc.

C Syntax

```
BOOL CRPE_API PESetParameterValueInfo(
        short printJob,
        const char FAR *parameterFieldName,
        const char FAR *reportname,
        PEParameterValueInfo FAR *valueInfo );
```

Parameters

printJob	Specifies the print job for which you want to set parameter value information.
parameterFieldName	Specifies a pointer to the string containing the parameter field name.
reportName	Specifies a pointer to the string containing the report name. See Remarks below.
valueInfo	Specifies a pointer to “PEParameterValueInfo” on page 488, which contains the parameter value information.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

Regarding parameter reportName:

- For the main report, pass an empty string ("").
- For a subreport, pass the file path and name of the subreport as a NULL-terminated string.

VB Syntax

```
Declare Function PESetParameterValueInfo Lib "crpe32.dll" ( _
    ByVal printJob As Integer, ByVal parameterFieldName As String, _
    ByVal reportName As String, valueInfo As PEParameterValueInfo ) As
Integer
```

Delphi Syntax

```
function PESetParameterValueInfo (
    printJob: smallint;
    const parameterFieldName: PChar;
    const reportName: PChar;
    var valueInfo: PEParameterValueInfo
    ): BOOL stdcall;
```

PESetPrintDate

Use PESetPrintDate to set a print date that may be different than the system calendar date. Use this function any time you want to show a print date (or use a print date in formulas) other than the actual date of printing.

C Syntax

```
BOOL CRPE_API PESetPrintDate (
        short printJob,
        short year,
        short month,
        short day );
```

Parameters

printJob	Specifies the print job for which you want to set the print date.
year	Specifies the year component of the print date. Enter a 4 digit year value (1994, 1993, etc.).
month	Specifies the month component of the print date. Months are numbered from 1 to 12, where January = 1 and December = 12. To use July as the print month, for example, you would enter the value 7.
day	Specifies the day component of the print date. Enter the actual day of the month you want to use (7, 18, 28, etc.).

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

- You change the print date, typically, when you want to run the report today yet have it appear to have been run on a different date. An example would be,

if you were out of town on the last day of the previous month and you later want to run a report for that month and make it appear as if it were run on the last day of the month.

- This function should be called before "PEStartPrintJob" on page 448, or the results may be inconsistent or unexpected.

VB Syntax

```
Declare Function PESetPrintDate Lib "crpe32.dll" (ByVal printJob As
Integer, ByVal Date_Year As Integer, ByVal Date_Month As Integer, ByVal
Date_Day As Integer) As Integer
```

Delphi Syntax

```
function PESetPrintDate (
    printJob: Word;
    year: smallint;
    month: smallint;
    day: smallint
    ): Bool stdcall;
```

dBASE for Windows Syntax

```
EXTERN CLOGICAL PESetPrintDate (CWORD, CWORD, CWORD, CWORD) CRPE.DLL
```

PESetPrintOptions

Use PESetPrintOptions to set the print options for the report to the values supplied in "PEPrintOptions" on page 489. Use this function any time you want to set the starting page number, the ending page number, the number of report copies, and / or collation instructions for a print job in response to user specifications at runtime.

C Syntax

```
BOOL CRPE_API PESetPrintOptions (
    short printJob,
    PEPrintOptions FAR *options );
```

Parameters

printJob	Specifies the print job for which you want to set printing options.
options	Specifies a pointer to "PEPrintOptions" on page 489. If this parameter is set to 0 (NULL), the function prompts the user for these options. Using this, you can get the behavior of the print-to-printer button in the preview window by calling PESetPrintOptions with a NULL pointer and then calling "PEPrintWindow" on page 387.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PESetPrintOptions Lib "crpe32.dll" (ByVal printJob As
Integer, Options As PEPrintOptions) As Integer
```

Delphi Syntax

```
function PESetPrintOptions (
    printJob: Word;
    var options: PEPrintOptions
    ): Bool stdcall;
```

PESetReportOptions

Use PESetReportOptions to set various options for the report to the values supplied in "PEReportAlertInfo" on page 494. Use this function any time you want to set any of the report options found in the Report | Options dialog box in the Crystal Reports Designer.

C Syntax

```
BOOL CRPE_API PESetReportOptions (
        short printJob,
        PEReportOptions FAR *reportOptions );
```

Parameters

printJob	Specifies the print job for which you want to set report options.
reportOptions	Specifies a pointer to "PEReportAlertInfo" on page 494.

Returns

- TRUE if the call is successful
- FALSE if the call fails.

VB Syntax

```
Declare Function PESetReportOptions Lib "crpe32.dll" (ByVal printJob _
        As Integer, reportOptions As PEReportOptions) As Integer
```

Delphi Syntax

```
function PESetReportOptions (
    printJob: smallint;
    var reportOptions: PEReportOptions
    ): Bool stdcall;
```

PESetReportSummaryInfo

Use PESetReportSummaryInfo to set report summary information. Report summary information corresponds to the Summary Info dialog box found in Crystal Reports.

C Syntax

```
BOOL CRPE_API PESetReportSummaryInfo (
        short printJob,
        PEReportSummaryInfo FAR *summaryInfo );
```

Parameters

printJob	Specifies the print job for which you want to set summary information.
summaryInfo	Specifies the pointer to “PEReportSummaryInfo” on page 499.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

This function should be called before “PEStartPrintJob” on page 448, or the results may be inconsistent or unexpected.

VB Syntax

```
Declare Function PESetReportSummaryInfo Lib “crpe32.dll” (
    ByVal printJob as Integer, summaryInfo as PEReportSummaryInfo ) as
Integer
```

Delphi Syntax

```
function PESetReportSummaryInfo (
    printJob: Word;
    var summaryInfo: PEReportSummaryInfo
    ): Bool; stdcall
```

PESetReportTitle

Use PESetReportTitle to change the report title in the report summary information. This function can also be used as one of a series of functions: “PEGetReportTitle” on page 358; “PEGetHandleString” on page 318; or REFSetReportTitle). This series can be used in a Custom-Print Link to identify and then change an existing report title in response to a user selection at print time.

C Syntax

```
BOOL CRPE_API PESetReportTitle (
        short printJob,
        const char FAR *title );
```

Parameters

printJob	Specifies the print job for which you want to set the report title.
title	Specifies a pointer to the null-terminated string containing the new title that you want to assign to the report.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

- When you give the user the ability to change the report title at print time, your link must include code to replace text with a user-generated value.
- This function should be called before "PEStartPrintJob" on page 448, or the results may be inconsistent or unexpected.

VB Syntax

```
Declare Function PESetReportTitle Lib "crpe32.dll" (
    ByVal printJob As Integer, ByVal Title As String ) As Integer
```

Delphi Syntax

```
function PESetReportTitle (
    printJob: Word;
    title: PChar
    ): Bool stdcall;
```

dBASE for Windows Syntax

```
EXTERN CLOGICAL PESetReportTitle (CWORD, CSTRING) CRPE.DLL
```

PESetSectionFormat

Use PESetSectionFormat to set the section format settings for selected sections in the specified report to the values in "PESectionOptions" on page 501. This function can be used to provide specialized formatting for printing invoices, form letters, printing to pre-printed forms, etc. It allows you to hide a section, insert a page break either before or after a section begins, reset the page number to 1 after a group value prints, prevent page breaks from spreading data from a single record over two pages, and to print group values only at the bottom of a page.

C Syntax

```
BOOL CRPE_API PESetSectionFormat (
        short printJob,
        short sectionCode,
        PESectionOptions FAR *options );
```

Parameters

printJob	Specifies the print job for which you want to set section formatting options.
sectionCode	Specifies the “Section Codes” on page 559 for the report section(s) for which you want to set formatting options. See “Working with section codes” on page 46.
options	Specifies a pointer to “PESectionOptions” on page 501. Use this structure to set your section options.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

- There can be multiple sections in an area.
- This function should be called before “PEStartPrintJob” on page 448 or the results may be inconsistent or unexpected.

VB Syntax

```
Declare Function PESetSectionFormat Lib “crpe32.dll” (
    ByVal printJob As Integer, ByVal sectionCode As Integer,
    Options As PESectionOptions ) As Integer
```

Delphi Syntax

```
function PESetSectionFormat (
    printJob: Word;
    sectionCode: smallint;
    var options: PESectionOptions
    ): Bool stdcall;
```

PESetSectionFormatFormula

Use PESetSectionFormatFormula to change the specified section format formula to the formula string you supply as a parameter. This function will only change the text of a formula which already exists in the report; you cannot use it to add a formula. When you give the user the ability to change the formula at print time, your link must include code to replace the formulaString parameter with a user-generated value.

C Syntax

```
BOOL CRPE_API PESetSectionFormatFormula (
        short printJob,
        short sectionCode,
        short formulaName,
        const char FAR *formulaString );
```

Parameters

printJob	Specifies the print job for which you want to set a new selection formula string.
sectionCode	Specifies the "Section Codes" on page 559, for the report section(s) for which you want to set formatting options. See "Working with section codes" on page 46.
formulaName	Specifies the name of the formatting formula for which you want to supply a new string. Use one of the PE_FFN_XXX "Area / Section Format Formula Constants" on page 541.
formulaString	Specifies a pointer to the null-terminated string that you want to assign to the format formula.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.
- Error code PE_ERR_BADFORMULANAME if the formula does not exist.
- Error code PE_ERR_BADFORMULATEXT if there is an error in the formula.

Remarks

- This function should be called before "PEStartPrintJob" on page 448, or the results may be inconsistent or unexpected.
- Not all formula names can be applied to all sections.

VB Syntax

```
Declare Function PESetSectionFormatFormula Lib "crpe32.dll" (
    ByVal printJob As Integer, ByVal sectionCode As Integer,
    ByVal formulaName As Integer, ByVal FormulaString As String ) As
Integer
```

Delphi Syntax

```
function PESetSectionFormatFormula (
    printJob: Word;
    sectionCode: smallint;
    formulaName: smallint;
    formulaString: PChar
    ): Bool stdcall;
```

PESetSectionHeight

Use PESetSectionHeight to set the section height information for the specified section. This is the replacement API Call for *PESetMinimumSectionHeight* and should be used for all new development.

C Syntax

```
BOOL CRPE_API PESetSectionHeight(
        short printJob,
        short sectionCode,
        short height );
```

Parameters

printJob	Specifies the print job for which you want to set section height information.
sectionCode	Specifies the "Section Codes" on page 559 for the section(s) for which you want to set section height information. See "Working with section codes" on page 46.
height	The section height measured in twips.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PESetSectionHeight Lib "crpe32.dll" (
    ByVal printJob As Integer, ByVal sectionCode As Integer,
    ByVal height As Integer ) As Integer
```

Delphi Syntax

```
function PESetSectionHeight (
    printJob: Smallint;
    sectionCode: Smallint;
    Height: Smallint (*in twips*)
    ): Bool stdcall;
```

PESetSelectionFormula

Use PESetSelectionFormula to change the selection formula to the formula string that you supply as a parameter. This function can be used by itself to replace a known record selection formula and also as one of a series of functions: "PEGetSelectionFormula" on page 365; "PEGetHandleString" on page 318; or PESetSelectionFormula). The series can be used in a Custom-Print link to identify and then change an existing record selection formula at print time in response to a user selection. When you give the user the ability to change the record selection formula at print time, your link must include code to replace formulaString with a user-generated value.

C Syntax

```
BOOL CRPE_API PESetSelectionFormula (
        short printJob,
        const char FAR *formulaString );
```

Parameters

printJob	Specifies the print job for which you want to set a new selection formula string.
formulaString	Specifies a pointer to the null-terminated string that you want to assign to the selection formula.

Returns

- TRUE if the call is successful.
- FALSE if the call fails due to an internal error (for example, the connection to the database fails).

Remarks

- Immediately after setting the new formula with PESetSelectionFormula, the new formula should be verified with “PECheckSelectionFormula” on page 285.
- This function should be called before “PEStartPrintJob” on page 448 or the results may be inconsistent or unexpected.

VB Syntax

```
Declare Function PESetSelectionFormula Lib “crpe32.dll” (ByVal printJob
As Integer, ByVal FormulaString As String) As Integer
```

Delphi Syntax

```
function PESetSelectionFormula (
    printJob: Word;
    formulaString: PChar
    ): Bool stdcall;
```

dBASE for Windows Syntax

```
EXTERN CLOGICAL PESetSelectionFormula (CWORD, CSTRING) CRPE.DLL
```

PESetSQLExpression

Use PESetSQLExpression to enter an SQL expression for a specified report.

C Syntax

```
BOOL CRPE_API PESetSQLExpression (
        short printJob,
        const char FAR *expressionName,
        const char FAR *expressionString );
```

Parameters

printJob	Specifies the print job for which you want to set an SQL expression.
expressionName	Specifies a pointer to the string containing the expression name.
expressionString	Specifies a pointer to the string containing the SQL expression.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PESetSQLExpression Lib "crpe32.dll" (ByVal printJob As
Integer, ByVal expressionName As String, ByVal expressionString As
String) As Integer
```

Delphi Syntax

```
function PESetSQLExpression (
    printJob: Smallint;
    const expressionName: PChar;
    const expressionString: PChar
    ): Bool stdcall;
```

PESetSQLQuery

Use PESetSQLQuery to change the SQL query to the query string you supply as a parameter. Use this function to update the SQL query that will be used to print the specified report, typically to add optimizations to the WHERE clause.

C Syntax

```
BOOL CRPE_API PESetSQLQuery (
        short printJob,
        const char FAR *queryString );
```

Parameters

printJob	Specifies the print job for which you want to modify the SQL query.
queryString	Specifies a pointer to the null-terminated string that you want to use to replace the existing SQL query.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

- This function is useful for reports with SQL queries that were explicitly edited in the Show SQL Query dialog box in Crystal Reports (that is, those reports that needed database-specific selection criteria or joins). Otherwise it is usually best to continue using function calls such as "PESetSelectionFormula" on page 441, and let Crystal Reports build the SQL query automatically.
- PESetSQLQuery has the same restrictions as editing in the Show SQL Query dialog box. In particular, changes are accepted in the WHERE and ORDER BY clauses but they are ignored in the SELECT list of fields.
- This call only applies to reports created against an ODBC source or on a native SQL database connection.

VB Syntax

```
Declare Function PESetSQLQuery Lib "crpe32.dll" ( ByVal printJob As Integer,
ByVal QueryString As String ) As Integer
```

Delphi Syntax

```
function PESetSQLQuery (
    printJob: Word;
    queryString: PChar
    ): Bool stdcall;
```

dBASE for Windows Syntax

```
EXTERN CLOGICAL PESetSQLQuery (CWORD, CSTRING) CRPE.DLL
```

PESetTrackCursorInfo

Use PESetTrackCursorInfo to set information for tracking the position of the mouse cursor over the preview window. This functions is only valid if the report was sent to a preview window (see "PEOutputToWindow" on page 382), and if events for the preview window have been enabled (see "PEEnableEvent" on page 297).

C Syntax

```
BOOL CRPE_API PESetTrackCursorInfo (
        short printJob,
        PETrackCursorInfo FAR *cursorInfo );
```

Parameters

printJob	Specifies the print job for which you want to track the mouse cursor.
cursorInfo	Specifies a pointer to "PETrackCursorInfo" on page 514.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

- By default, all area's track cursors are arrow cursors. If the preview window contains a drill-down field (database field, summary, group name, group graph), the group area will use a magnify cursor. Group graphs in other areas will also use the magnify cursor.
- This function can be used to give the user visual feed back especially when tracking events.

VB Syntax

```
Declare Function PESetTrackCursorInfo Lib "crpe.dll" (ByVal printJob As
Integer, cursorInfo As PETrackCursorInfo) As Integer
```

Delphi Syntax

```
function PESetTrackCursorInfo(
    printJob: smallint;
    var cursorInfo: PETrackCursorInfo
    ): smallint stdcall;
```

PESetWindowOptions

Use PESetWindowOptions to set display options for the preview window, including which preview window controls are available. This function must be called after "PEOutputToWindow" on page 382.

C Syntax

```
BOOL CRPE_API PESetWindowOptions (
        short printJob,
        PEWindowOptions FAR *options );
```

Parameters

printJob	Specifies the print job for which you want to set preview window display options.
options	Specifies a pointer to a "PEWindowOptions" on page 519.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PESetWindowOptions Lib "crpe.dll" (
    ByVal printJob As Integer, Options As PEWindowOptions ) As Integer
```

Delphi Syntax

```
function PESetWindowOptions (
    printJob: Word;
    var options: PEWindowOptions
    ): Bool stdcall;
```

PEShow...Page

Use the PEShow...Page function calls to display the specified page in the preview window. Use these functions any time you want to display specific pages of a report in the preview window or give the user the ability to move forward, backward, or to a specified page in a report in the preview window. Use "PEGetNPages" on page 324, to determine the number of pages available in the specified report when using PEShowNthPage, for example.

C Syntax

```
BOOL CRPE_API PEShowFirstPage (
        short printJob );
BOOL CRPE_API PEShowLastPage (
        short printJob );
BOOL CRPE_API PEShowNextPage (
        short printJob );
BOOL CRPE_API PEShowNthPage (
        short printJob,
        short pageN );
BOOL CRPE_API PEShowPreviousPage (
        short printJob );
```

Parameter(s)

printJob	Specifies the print job for which you want to indicate the page to be displayed.

- The following parameter applies only to PEShowNthPage.

pageN	Specifies the 1-based number indicating the report page that you want to display when calling PEShowNthPage. See Remarks below.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

Using “PEGetJobStatus” on page 319, you can determine how many pages are contained in the specified report. This information will determine the range available for parameter pageN when using PEShowNthPage.

VB Syntax

```
Declare Function PEShowFirstPage Lib “crpe32.dll” (
    ByVal printJob As Integer ) As Integer
Declare Function PEShowLastPage Lib “crpe32.dll” (
    ByVal printJob As Integer ) As Integer
Declare Function PEShowNextPage Lib “crpe32.dll” (
    ByVal printJob As Integer ) As Integer
Declare Function PEShowNthPage Lib “crpe32.dll” (
    ByVal printJob As Integer, ByVal pageN As Integer ) As Integer
Declare Function PEShowPreviousPage Lib “crpe32.dll” (
    ByVal printJob As Integer ) As Integer
```

Delphi Syntax

```
function PEShowFirstPage (
    printJob: Word
    ): Bool stdcall;
function PEShowLastPage (
    printJob: Word
    ): Bool stdcall;
function PEShowNextPage (
    printJob: Word
    ): Bool stdcall;
function PEShowPreviousPage (
    printJob: Word
    ): Bool stdcall;
function PEShowNthPage (
    printJob: Word;
    pageN: Smallint
    ): Bool stdcall;
```

dBASE for Windows Syntax

```
EXTERN CLOGICAL PEShowNextPage (CWORD) CRPE.DLL
EXTERN CLOGICAL PEShowFirstPage (CWORD) CRPE.DLL
EXTERN CLOGICAL PEShowPreviousPage (CWORD) CRPE.DLL
EXTERN CLOGICAL PEShowLastPage (CWORD) CRPE.DLL
```

PEShowPrintControls

Use PEShowPrintControls to display the print controls (First, Previous, Next, and Last Page buttons as well as the buttons for Cancel, Close, Export, and Print to Printer). Use this call any time you want to provide control over whether print controls are displayed or not.

C Syntax

```
BOOL CRPE_API PEShowPrintControls (
        short printJob,
        BOOL showPrintControls );
```

Parameters

printJob	Specifies the print job for which you want to toggle the display of print controls.
showPrint Controls	Boolean value indicates whether or not the print controls are displayed. TRUE will cause the print controls to be displayed; FALSE will hide the print controls.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

Print controls are displayed by default. It is not necessary to use this function simply to display controls but only if you want the user to control whether or not the controls are visible.

VB Syntax

```
Declare Function PEShowPrintControls Lib "crpe32.dll" (
    ByVal printJob As Integer, ByVal ShowPrintControls As Integer ) As
Integer
```

Delphi Syntax

```
function PEShowPrintControls (
    printJob: Word;
    showPrintControls: Bool
    ): Bool stdcall;
```

dBASE for Windows Syntax

```
EXTERN CLOGICAL PEShowPrintControls (CWORD, CLOGICAL) CRPE.DLL
```

PEStartPrintJob

Use PEStartPrintJob to start the printing of a report. This function is used as a mandatory part of each Custom-Print Link to trigger the printing of a report to the printer or to the preview window.

C Syntax

```
BOOL CRPE_API PEStartPrintJob (
        short printJob,
        BOOL waitUntilDone );
```

Parameters

printJob	Specifies the print job you want to start.
waitUntilDone	BOOL. Reserved. This parameter must always be set to TRUE.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

Remarks

A print job can be started only once. Once started, the only function that can be used is “PEClosePrintJob” on page 288.

VB Syntax

```
Declare Function PEStartPrintJob Lib “crpe32.dll” (ByVal printJob _
        As Integer, ByVal WaitOrNot As Integer) As Integer
```

Delphi Syntax

```
function PEStartPrintJob (
    printJob: Word;
    waitUntilDone: Bool
    ): Bool stdcall;
```

dBASE for Windows Syntax

```
EXTERN CLOGICAL PEStartPrintJob (CWORD, CLOGICAL) CRPE.DLL
```

PETestNthTableConnectivity

Use PETestNthTableConnectivity to test whether a database table's settings are valid and ready to be reported on. This function is typically used if you plan to print at a later time but you want to test now to make sure everything is in order for logging on. For example, your application can prompt your user for a password and then test it before printing begins. See Remarks below.

C Syntax

```
BOOL CRPE_API PETestNthTableConnectivity (
        short printJob,
        short tableN );
```

Parameters

printJob	Specifies the print job for which you want to test a table’s connection settings.
tableN	Specifies the 0-based number of the table for which you want to test the database table settings. The first table is table 0. The last table is N-1.

Returns

- TRUE if the database session, log on, and location information is all correct.
- FALSE if the call fails.

Remarks

- This function may require a significant amount of time to complete, since it will first open a user session (if required), then log on to the database server (if required), and then open the appropriate database table (to test that it exists).
- PETestNthTableConnectivity does not read any data.
- PETestNthTableConnectivity closes the table immediately once the connection has been tested successfully.
- If the connection fails at one of the following steps, the error code set indicates which function needs to be called to provide updated database information.

Connection Failure	Error Code	Update with Function
Unable to begin a session.	PE_ERR_DATABASESESSION	"PESetNthTableSessionInfo" on page 429
Unable to log on to a server.	PE_ERR_DATABASELOGON	"PESetNthTableLogOnInfo" on page 427
Unable to open the table.	PE_ERR_DATABASELOCATION	"PESetNthTableLocation" on page 426

- Logging off is performed when the print job is closed.

VB Syntax

```
Declare Function PETestNthTableConnectivity Lib "crpe32.dll" ( ByVal
printJob As Integer, ByVal TableN As Integer ) As Integer
```

Delphi Syntax

```
function PETestNthTableConnectivity (
    printJob: Word;
    tableN: smallint
    ): Bool stdcall;
```

dBASE for Windows Syntax

```
EXTERN CLOGICAL PETestNthTableConnectivity (CWORD, CWORD) CRPE.DLL
```

PEVerifyDatabase

Use PEVerifyDatabase to verify that a connection to the database(s) specified in a report are valid.

C Syntax

```
BOOL CRPE_API PEVerifyDatabase (
        short printJob );
```

Parameter

printJob	Specifies the print job for which you want to verify the database connection.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEVerifyDatabase Lib "crpe32.dll" (
    ByVal printJob As Integer ) As Integer
```

Delphi Syntax

```
function PEVerifyDatabase
    printJob: smallint
    ): Bool stdcall;
```

PEZoomPreviewWindow

Use PEZoomPreviewWindow to change the magnification of the preview window to a specified level. Use this function when the report has been printed to a preview window and you want to set the magnification of the preview window to a specific level (Full Page, Fit One Side, Fit Both Sides, or a specified level).

C Syntax

```
BOOL CRPE_API PEZoomPreviewWindow (
        short printJob,
        short level );
```

Parameters

printJob	Specifies the print job for which you want to set the magnification level in the preview window.
level	The zoom level at which you want to set the preview window. This value can be a set to a specific magnification level (%) in the range from 25 to 400, or you can use one of the “Zoom Level Constants” on page 561.

Returns

- TRUE if the call is successful.
- FALSE if the call fails.

VB Syntax

```
Declare Function PEZoomPreviewWindow Lib "crpe32.dll" (
    ByVal printJob As Integer, ByVal ZoomLevel As Integer ) As Integer
```

Delphi Syntax

```
function PEZoomPreviewWindow (
    printJob: Word;
    level: smallint
    ): Bool stdcall;
```

dBASE for Windows Syntax

EXTERN CLOGICAL PEZoomPreviewWindow (CWORD, CWORD)CRPE.DLL

Print Engine Structures

The print Engine structures are listed alphabetically in this section.

PEAlertInstanceInfo

PEAlertInstanceInfo contains information for a selected instance of a Report Alert. This information is used by “PEGetNthAlertInstanceInfo” on page 333, to retrieve information on the selected instance of the Report Alert.

C Syntax

```
typedef struct PEAlertInstanceInfo
{
    WORD StructSize;
    short alertMessageLength;
    HANDLE alertMessage;
} PEAlertInstanceInfo;
```

Members

StructSize	Specifies the size of the PEAlertInstanceInfo structure. Initialize this member to PE_SIZEOF_ALERT_INSTANCE_INFO.
alsertMessage Length	Specifies the length of the alert message generated for this instance of the Report Alert.
alertMessage	Specifies a handle to the string containing alert message generated for this instance of the Report Alert.

Remarks

In the present build only the first instance of the Report Alert is created at runtime. This limitation will be addressed in a future release.

VB Type Listing

```
Type PEAlertInstanceInfo
    StructSize As Integer
    alertMessageLength As Integer
    alertMessage As Long
End Type
```

Delphi Record Listing

```
type PEAlertInstanceInfo = record
    StructSize : Word;
    alertMessageLength : Smallint;
    alertMessage : HWND;
end;
```

PECloseButtonClickedEventInfo

This structure contains information about a close button clicked event. When the close button in a preview window is clicked, a callback function will be called with EventId = PE_CLOSE_BUTTON_CLICKED_EVENT.

C Syntax

```
typedef struct PECloseButtonClickedEventInfo {
    WORD StructSize;
    WORD viewIndex;
    long windowHandle;
} PECloseButtonClickedEventInfo;
```

Members

StructSize	Specifies the size of the PECloseButtonClickedEventInfo structure. Initialize this number to PE_SIZEOF_CLOSE_BUTTON_CLICKED_EVENT_INFO.
viewIndex	Specifies the 0-based index number indicating which view is going to be closed.
windowHandle	Specifies the handle for the frame window on which the button is placed.

VB Type Listing

```
Type PECloseButtonClickedEventInfo
    StructSize As Integer
    viewIndex As Integer
    windowHandle As Long
End Type
```

Delphi Record Listing

```
type
    PECloseButtonClickedEventInfo = record
        StructSize: Word;
        viewIndex: Word;
        windowHandle: HWnd;
    end;
```

PEDrillOnDetailEventInfo

PEDrillOnDetailEventInfo contains information related to callback Event Id = PE_DRILL_ON_DETAIL_EVENT event information.

C Syntax

```
typedef struct PEDrillOnDetailEventInfo {
    WORD StructSize;
    short selectedFieldIndex;
    long windowHandle;
    struct PEFieldValueInfo **fieldValueList;
    short nFieldValue;
} PEDrillOnDetailEventInfo;
```

Members

StructSize	Specifies the size of the PEDrillOnDetailEventInfo structure. Initialize this number to PE_SIZEOF_DRILL_ON_DETAIL_EVENT_INFO.
selectedFieldIndex	The 0-based index indicating which drill-down field was selected. Contains -1 if no field was selected.
windowHandle	Frame window handle where the drill on detail event happens.
fieldValueList	Points to an array of PEFieldValue. Memory pointed by fieldValueList is freed after calling the callback function.
nFieldValue	The 1-based index of the value in field value list (for example, if the value is listed second, the number would be 2).

Remarks

If the user clicks one of the fields in the Details section, selectedFieldIndex will point to the field index in fieldValueList. These fields have to be one of database field, group name field, summary field, formula field. Clicks on text object, graph, picture, ole, subreport, special var field, or database memo or blob field, selectedFieldIndex return -1.

Delphi Record Listing

```
type
    PEFieldValueInfoDoublePtr = ^PEFieldValueInfoPtr
    PEDrillOnDetailEventInfo = record{
        StructSize: Word;
        selectedFieldIndex:smallint;
        windowHandle: longint;
        fieldValueList: PEFieldValueInfoDoublePtr;
        nFieldValue: smallint;
    end;
```

PEDrillOnGroupEventInfo

PEDrillOnGroupEventInfo specifies drill on group information when PE_DRILL_ON_GROUP_EVENT happens.

C Syntax

```
typedef struct PEDrillOnGroupEventInfo {
    WORD StructSize;
    WORD drillType;
    long windowHandle;
    char **groupList;
    WORD groupLevel;
} PEDrillOnGroupEventInfo;
```

Members

<table>
<tr><td>StructSize</td><td colspan="2">Specifies the size of the PEDrillOnGroupEventInfo structure. Initialize this number to PE_SIZEOF_DRILL_ON_GROUP_EVENT_INFO.</td></tr>
<tr><td>drillType</td><td colspan="2">Specifies the type of drill down that is used. Use one of the following values.</td></tr>
<tr><td></td><td>Constant</td><td>Description</td></tr>
<tr><td></td><td>PE_DE_ON_GROUP</td><td>Drill-down on a group summary or subtotal.</td></tr>
<tr><td></td><td>PE_DE_ON_GROUPTREE</td><td>Drill-down a Group Tree node.</td></tr>
<tr><td></td><td>PE_DE_ON_GRAPH</td><td>Drill-down a group graph object.</td></tr>
<tr><td></td><td>PE_DE_ON_MAP</td><td>Drill-down on a map region.</td></tr>
<tr><td></td><td>PE_DE_ON_SUBREPORT</td><td>Drill-down on a subreport.</td></tr>
<tr><td>windowHandle</td><td colspan="2">Frame window handle where the event happens.</td></tr>
<tr><td>groupList</td><td colspan="2">Specifies an array of pointers to group names in the report when drilling on a group summary, a Group Tree, a chart, or a map; and a pointer to a single element array containing the subreport name when drilling on a subreport. This memory is freed after the callback function is called.</td></tr>
<tr><td>groupLevel</td><td colspan="2">The number of the group name in the group list.</td></tr>
</table>

Remarks

Member groupList will be freed after the callback function. You will need to make a copy of the groupList if you want to use it later.

VB Type Listing

```
Type PEDrillOnGroupEventInfo
    StructSize As Integer
    drillType As Integer
    windowHandle As Long
    groupList As String
    groupLevel As Integer
End Type
```

Delphi Record Listing

```
type
    PEPCharPointer = ^PChar
    PEDrillOnGroupEventInfo
        StructSize: Word;
        drillType: Word;
        windowHandle: HWnd;
        groupList: PEPCharPointer;
        groupLevel: Word;
    end;
```

PEEnableEventInfo

Events are grouped in CRPE. PEEnableEventInfo specifies which group event is enabled or disabled. All events are disabled by default. Use "PEEnableEvent" on page 297, to enable events.

C Syntax

```
typedef struct PEEnableEventInfo {
    WORD StructSize;
    short startStopEvent;
    short readingRecordEvent;
    short printWindowButtonEvent;
    short drillEvent;
    short closePrintWindowEvent;
    short activatePrintWindowEvent;
    short fieldMappingEvent;
    short mouseClickEvent;
    short hyperlinkEvent;
    short launchSeagateAnalysisEvent;
} PEEnableEventInfo;
```

Members

Each member of type short can be set to TRUE, FALSE, or PE_UNCHANGED for no change.

StructSize	Specifies the size of the PEEnableEventInfo structure. Initialize this number to PE_SIZEOF_ENABLE_EVENT_INFO.
startStopEvent	Boolean value, or PE_UNCHANGED for no change. Start/Stop event.
readingRecordEvent	Boolean value, or PE_UNCHANGED for no change. Reading record event.
printWindowButton Event	Boolean value, or PE_UNCHANGED for no change. Print window button event.
drillEvent	Boolean value, or PE_UNCHANGED for no change. Drill event.
closePrintWindowEvent	Boolean value, or PE_UNCHANGED for no change. Close print window event.
activatePrintWindow Event	Boolean value, or PE_UNCHANGED for no change. Activate print window event.
fieldMappingEvent	Boolean value, or PE_UNCHANGED for no change. Field mapping event.
mouseClickEvent	Boolean value, or PE_UNCHANGED for no change. Mouse click event.
hyperlinkEvent	Boolean value, or PE_UNCHANGED for no change. Hyperlink event.
launchSeagateAnalysis Event	Boolean value, or PE_UNCHANGED for no change. Launch Seagate Analysis event. **Note:** Seagate Analysis is now known as Crystal Analysis.

Remarks

- By default, all events are disabled. Use “PEEnableEvent” on page 297, to enable the desired event.
- For startStopEvent, readingRecordEvent, printWindowEvent, drillEvent, closePrintWindowEvent, and activatePrintWindowEvent, use PE_UNCHANGED for no change.

VB Type Listing

```
Type PEEnableEventInfo
    StructSize As Integer
    startStopEngine As Integer
    readingRecordEvent As Integer
    printWindowButtonEvent As Integer
    drillEvent As Integer
    closePrintWindowEvent As Integer
    activatePrintWindowEvent As Integer
    fieldMappingEvent As Integer
    mouseClickEvent As Integer
End Type
```

Delphi Record Listing

```
type
    PEEnableEventInfo = record
        StructSize: Word;
        startStopEvent: smallint;
        readingRecordEvent: smallint;
        printWindowButtonEvent: smallint;
        drillEvent: smallint;
        closePrintWindowEvent: smallint;
        activatePrintWindowEvent: smallint;
        fieldMappingEvent: smallint;
        mouseClickEvent: smallint;
    end;
```

PEExportOptions

PEExportOptions contains file format and output destination information that is retrieved by “PEGetExportOptions” on page 306, and used “PEExportTo” on page 299, when exporting reports.

C Syntax

```
typedef struct PEExportOptions {
    WORD StructSize;
    char formatDLLName [PE_DLL_NAME_LEN];
    DWORD formatType;
    void FAR *formatOptions;
    char destinationDLLName [PE_DLL_NAME_LEN];
    DWORD destinationType;
    void FAR *destinationOptions;
    WORD nFormatOptionsBytes;
    WORD nDestinationOptionsBytes;
} PEExportOptions;
```

Members

Visual Basic developers should refer to the VB syntax for specifics of the VB structure.

StructSize	Specifies the size of the PEExportOptions structure. Initialize the member to PE_SIZEOF_EXPORT_OPTIONS.	
formatDLLName	Specifies a pointer to the null-terminated string that contains the format DLL name (of length PE_DLL_NAME_LEN = 64). The DLL is selected based on the format in which you want to export your report. Select the appropriate DLL name from the table below.	
	To export in this format	**Use this DLL**
	Crystal Reports Format	u2fcr.dll
	Data Interchange Format	u2fwordw.dll
formatType	Specifies the type of format you want to use from those types supported by the selected DLL. Whether the format DLL you select supports only one format type (for example, uxfcr.dll) or multiple format types (for example, uxfdoc.dll), you must still fill in this member. Select the format type you want to use from the table below.	
	To export a report in this format	**Use this formatType**
	Crystal Reports Format	UXFCrystalReportType
	Data Interchange Format	UXFDIFType
	Word for Windows Format	UXFWordWinType
	Word for DOS Format	UXFWordDosType
	WordPerfect Format	UXFWordPerfectType
	Quattro Pro 5.0 (WB1) Format	UXFQP5Type
	Record Style Format (column of values)	UXFRecordType
	Rich Text Format	UXFRichTextFormatType
	Comma Separated Values Format (CSV)	UXFCommaSeparatedType
	Tab Separated Values Format	UXFTabSeparatedType
	Character Separated Values Format	UXFCharSeparatedType
	Text Format (ASCII)	UXFTextType
	Paginated Text Format (ASCII)	UXFPaginatedTextType
	Tab Separated Text Format	UXFTabbedTextType
	Lotus 1-2-3 (WK3)	UXFLotusWk3Type
	Excel 4.0	UXFXls4Type
	Excel 5.0	UXFXls5Type
	Excel 5.0 Tabular	UXFXlsTypeTab
	ODBC	UXFODBCType

	HTML	UXFHTML3Type
	Microsoft Internet Explorer 2 HTML	UXFExplorer2Type
	Netscape 2 HTML	UXFNetscape2Type
formatOptions	Specifies a pointer to a structure that supplies date and number information. This information is used by the PEExportOptions structure when you want to export in one of the formats that support date and number options and you want to hard code your options. Select the appropriate structure (if needed) from the table below. WARNING: This member must be pointing to a valid address until "PEStartPrintJob" on page 448, is called.	
	To export a report in this format	**Use this structure if you want to hard code formatOptions**
	Data Interchange Format	"UXFDIFOptions" on page 528
	Record Style Format (column of values)	"UXFRecordStyleOptions" on page 531
	Comma Separated Values (CSV)	"UXFCommaTabSeparatedOptions" on page 528
	Tab Separated Values	"UXFCommaTabSeparatedOptions" on page 528
	Character Separated Values	"UXFCharSeparatedOptions" on page 527
	Paginated Text	"UXFPaginatedTextOptions" on page 530
	Excel (Tabular)	
	ODBC Format	"UXFODBCOptions" on page 530
	HTML Format	"UXFHTML3Options" on page 529
destinationDLL Name	Specifies a pointer to the string (of length PE_DLL_NAME_LEN = 64, NULL-terminated) that contains the destination DLL name. The DLL used is determined by the destination to which you want to export your report. Select the appropriate DLL name from the table below.	
	To export a report to this destination	**Use this DLL name**	**Use this DLL name**
	Disk File	uxddisk.dll	u2ddisk.dll
	E-Mail (MAPI)	uxdmapi.dll	u2dmapi.dll
	E-Mail (VIM)	uxdvim.dll	u2dvim.dll
	Microsoft Exchange	uxdpost.dll	u2dpost.dll
destinationType	Specifies the type of destination you want to use from those types supported by the selected DLL. Even if the destinationDLL name you select supports only one destination type, you must still fill in this member. Select the destination type you want to use from the table below.	
	To export a report to this destination	**Use this destinationType**
	Disk File	UXDDiskType

	E-Mail (MAPI)	UXDMAPIType
	E-Mail (VIM)	UXDVIMType
	Microsoft Exchange	UXDExchFolderType
destination Options	Specifies a pointer to a structure containing information used by the PEExportOptions structure. This information is needed to export a report and hard code the file name (when exporting to Disk File) or e-mail message information (when exporting to MAPI or VIM destination). Select the appropriate structure (if needed) from the table below. WARNING: This member must be pointing to a valid address until "PEStartPrintJob" on page 448, is called.	
	To export a report to this destination	**Use this structure if you want to hard code destinationOptions**
	Disk File	"UXDDiskOptions" on page 522
	E-Mail (MAPI)	"UXDMAPIOptions" on page 523
	E-Mail (VIM)	"UXDVIMOptions" on page 526
	Microsoft Exchange	"UXDPostFolderOptions" on page 525
nFormat OptionsBytes	Set by "PEGetExportOptions" on page 306, and ignored by "PEExportTo" on page 299.	
nDestination Options-Bytes	Set by "PEGetExportOptions" on page 306, and ignored by "PEExportTo" on page 299.	

Remarks

Note that both the formatOptions and destinationOptions members must be pointing to a valid address until "PEStartPrintJob" on page 448, is called.

VB Type Listing

```
Type PEExportOptions
    StructSize As Integer
    FormatDLLName As String * PE_DLL_NAME_LEN
    FormatType1 As Integer
    FormatType2 As Integer
    FormatOptions1 As Integer
    FormatOptions2 As Integer
    DestinationDLLName As String * PE_DLL_NAME_LEN
    DestinationType1 As Integer
    DestinationType2 As Integer
    DestinationOptions1 As Integer
    DestinationOptions2 As Integer
    NFormatOptionsBytes As Integer
    NDestinationOptionsBytes As Integer
End Type
```

Delphi Record Listing

```
type
    PEDllNameType = array[0..PE_DLL_NAME_LEN-1] or Char;
    PEExportOptions = record
        StructSize: Word;
        formatDLLName: PEDllNameType;
        formatType: dWord;
        formatOptions: Pointer;
        destinationDLLName: PEDllNameType;
        destinationType: dWord;
        destinationOptions: Pointer;
        nFormatOptionsBytes: Word;
        nDestinationOptionsBytes: Word;
    end;
```

PEFieldMappingEventInfo

PEFieldMappingEventInfo contains information related to mapped database fields.

C Syntax

```
typedef struct PEFieldMappingEventInfo {
    WORD StructSize;
    PEReportFieldMappingInfo **reportFields;
    WORD nReportFields;
    PEReportFieldMappingInfo **databaseFields;
    WORD nDatabaseFields
} PEFieldMappingEventInfo;
```

Members

StructSize	Specifies the size of the PEFieldMappingEventInfo structure. Initialize the member to PE_SIZEOF_FIELDMAPPING_EVENT_INFO.
reportFields	A pointer to an array of pointers to “PEReportFieldMappingInfo” on page 492, containing information about fields in the report.
nReportFields	Size of the reportFields array (equivalent to the number of fields in the report).
databaseFields	A pointer to an array of pointers to “PEReportFieldMappingInfo” on page 492 data members containing information about fields in the new database file.
nDatabaseFields	Size of the databaseField array.

Remarks

To map a report field to a database field the member mappingTo of each “PEReportFieldMappingInfo” on page 492, in the member reportFields array is assigned the index of the appropriate field in the member databaseFields array.

Delphi Record Listing

```
type
       PEFieldMappingInfoPtr = ^PEReportFieldMappingInfo;
       PEFieldMappingInfoDoublePtr = ^PEFieldMappingInfoPtr;
 type
        PEFieldMappingEventInfo = record
         StructSize : Word;
         reportFields : PEFieldMappingInfoDoublePtr;
         nReportFields : Word;
         databaseFields : PEFieldMappingInfoDoublePtr;
         nDatabaseFields : Word;
  end;
```

PEFieldValueInfo

Specifies a field value in the fieldValueList of “PEDrillOnDetailEventInfo” on page 454.

C Syntax

```
typedef struct PEFieldValueInfo {
    WORD StructSize;
    WORD ignored;
    char fieldName [PE_FIELD_NAME_LEN];
    PEValueInfo fieldValue;
} PEFieldValueInfo;
```

Members

StructSize	Specifies the size of the PEFieldValueInfo structure. Initialize the member to PE_SIZEOF_FIELD_VALUE_INFO.
ignored	For 4 byte alignment. Ignore.
fieldName	Specifies the formula form of a field name (of length PE_FIELD_NAME_LEN = 512).
fieldValue	Specifies the field value for the selected Details area for the field specified by the field name.

Remarks

A selected Details area corresponds to a database record.

VB Type Listing

```
Type PEFieldValueInfo
    StructSize As Integer
    ignored As Integer
    fieldName As String
    fieldValue As PEValueInfo
End Type
```

Delphi Record Listing

```
type
    PEFieldNameType = array[0..PE_FIELD_NAME_LEN-1] of char;
    PEFieldValueInfo = record
        StructSize: Word;
        ignored: Word;
        fieldName: PEFieldNameType;
        fieldValue: PEValueInfo;
    end;
```

PEFontColorInfo

PEFontColorInfo contains information regarding chart title text fonts.

C Syntax

```
typedef struct PEFontColorInfo {
    WORD StructSize;
    char faceName[PE_FACE_NAME_LEN]; // empty string for no change
    short fontFamily;
    short fontPitch;
    short charSet;
    short pointSize;
    short isItalic;
    short isUnderlined;
    short isStruckOut;
    short weight;
    COLORREF color;
    short twipSize;
} PEFontColorInfo;
```

Members

StructSize	Specifies the size of the PEFontColorInfo structure. Initialize this member to PE_SIZEOF_FONT_COLOR_INFO.
faceName	Specifies the actual face name of the font [of length PE_FACE_NAME_LEN = 64]. The face name can typically come from a Font dialog box, be hard coded in the application or be chosen by the application from the fonts supported on the printer. For example, “Times New Roman”. Pass an empty string ("") for no change.
fontFamily	Specifies the font family for the font. Use one of the following FF_XXX Font Family Constants.

Constant	Description
FF_DONTCARE	No change.
FF_ROMAN	Variable pitch font with serifs.
FF_SWISS	Fixed pitch font without serifs.
FF_MODERN	Fixed-pitch font, with or without serifs.

	FF_SCRIPT	Handwriting-like font.
	FF_DECORATIVE	Fancy display font.
fontPitch	Specifies the font pitch. Use a constant value for the font pitch as defined in WINDOWS.H. Use DEFAULT_PITCH for the current default setting.	
charSet	Specifies the character set. Use a constant value for the character set as defined in WINDOWS.H. Use DEFAULT_CHARSET for the current default setting.	
pointSize	Specifies the desired point size for the selected font. Use this member or member twipSize to specify the font size. If both members are non-zero, then this member will be ignored and twipSize will be used. Pass 0 for both twipSize and pointSize for no change.	
isItalic	Specifies whether the font selected should be italicized. Use TRUE for Italic font, FALSE for non-Italic font, or PE_UNCHANGED for the current default setting.	
isUnderlined	Specifies whether the font selected should be underlined. Use TRUE for Underline, FALSE for no Underline, or PE_UNCHANGED for the current default setting.	
isStruckOut	Specifies whether the font selected should be struck out. Use TRUE for StruckOut, FALSE for no StruckOut, or PE_UNCHANGED for the current default setting.	
weight	Specifies the weight of the font. Use a constant value from the weight values defined in WINDOWS.H. Use 0 for no change.	
color	Specifies the RGB color value contained in “COLORREF” on page 533. Use PE_UNCHANGED_COLOR for the current default setting.	
twipSize	Specifies the font size, in twips. Use this member or member pointSize to specify the font size. If both members are non-zero, then this member will be used and pointSize will be ignored. Pass 0 for both twipSize and pointSize for no change.	

VB Type Listing

```
Type PEFontColorInfo
    StructSize As Integer
    faceName As String * PE_FACE_NAME_LEN
    fontFamily As Integer
    fontPitch As Integer
    charSet As Integer
    pointSize As Integer
    isItalic As Integer
    isUnderlined As Integer
    isStruckOut As Integer
    weight As Integer
    color As Long
    twipSize As Integer
End Type
```

Delphi Record Listing

```
type
  PEFaceNameType = array [0..PE_FACE_NAME_LEN-1] of Char;
  PEFontColorInfo = record

    StructSize   : Word;
    faceName     : PEFaceNameType;
    fontFamily   : Smallint;
    fontPitch    : Smallint;
    charSet      : Smallint;
    pointSize    : Smallint;
    isItalic     : Smallint;
    isUnderlined : Smallint;
    isStruckOut  : Smallint;
    weight       : Smallint;
    color        : COLORREF;
  end;
```

PEFormulaSyntax

PEFormulaSyntax is used by "PEGetFormulaSyntax" on page 308, and "PESetFormulaSyntax" on page 406, to retrieve and set the syntax for the current and following formula API calls.

C Syntax

```
#define PE_FS_SIZE 2
typedef struct PEFormulaSyntax {
    WORD StructSize;
    short formulaSyntax [PE_FS_SIZE];
} PEFormulaSyntax;
```

Members

StructSize	Specifies the size of the `PEFormulaSyntax` structure. Initialize this member to PE_SIZEOF_FORMULA_SYNTAX.
formulaSyntax	Specifies the formula syntax. Use one of the PE_FST_XXX "Formula Syntax Constants" on page 551, or PE_UNCHANGED for no change. Default value is PE_FST_CRYSTAL.

VB Syntax

```
Type PEFormulaSyntax
    StructSize As Integer
    formulaSyntax (0 To 1) As Integer
End Type
```

PEGeneralPrintWindowEventInfo

PEGeneralPrintWindowEventInfo contains general preview window event information. This structure is used for multiple events listed below. See Remarks.

C Syntax

```
typedef struct PEGeneralPrintWindowEventInfo {
    WORD StructSize;
    WORD ignored
    long windowHandle;
} PEGeneralPrintWindowEventInfo;
```

Members

StructSize	Specifies the size of the PEGeneralPrintWindowEventInfo structure. Initialize this member to PE_SIZEOF_GENERAL_PRINT_WINDOW_EVENT_INFO.
ignored	For 4 byte alignment. Ignore.
windowHandle	Frame window handle where the event happens.

Remarks

Structure PEGeneralPrintWindowEventInfo is used for the following events.

- PE_CLOSE_PRINT_WINDOW_EVENT
- PE_PRINT_BUTTON_CLICKED_EVENT
- PE_EXPORT_BUTTON_CLICKED_EVENT
- PE_FIRST_PAGE_BUTTON_CLICKED_EVENT
- PE_PREVIOUS_PAGE_BUTTON_CLICKED_EVENT
- PE_NEXT_PAGE_BUTTON_CLICKED_EVENT
- PE_LAST_PAGE_BUTTON_CLICKED_EVENT
- PE_CANCEL_BUTTON_CLICKED_EVENT
- PE_PRINT_SETUP_BUTTON_CLICKED_EVENT
- PE_REFRESH_BUTTON_CLICKED_EVENT
- PE_ACTIVATE_PRINT_WINDOW_EVENT
- PE_DEACTIVATE_PRINT_WINDOW_EVENT

VB Type Listing

```
Type PEGeneralPrintWindowEventInfo
    StructSize As Integer
    ignored As Integer
    windowHandle As Long
End Type
```

Delphi Record Listing

```
type
    PEGeneralPrintWindowEventInfo = record
        StructSize: Word;
        ignored: Word;
        windowHandle: HWnd;
    end;
```

PEGraphAxisInfo

PEGraphAxisInfo contains information about the gridline options, data ranges and formats and axis division features for the specified chart.

C Syntax

```
//axis division method
#define PE_ADM_AUTOMATIC                0
#define PE_ADM_MANUAL                   1
typedef struct PEGraphAxisInfo {
    WORD StructSize;
    short groupAxisGridLine;
    short dataAxisYGridLine;
    short dataAxisY2GridLine;
    short seriesAxisGridline;
    double dataAxisYMinValue;
    double dataAxisYMaxValue;
    double dataAxisY2MinValue;
    double dataAxisY2MaxValue;
    double seriesAxisMinValue;
    double seriesAxisMaxValue;
    short dataAxisYNumberFormat;
    short dataAxisY2NumberFormat;
    short seriesAxisNumberFormat;
    short dataAxisYAutoRange;
    short dataAxisY2AutoRange;
    short seriesAxisAutoRange;
    short dataAxisYAutomaticDivision;
//PE_ADM_* or PE_UNCHANGED for no change
    short dataAxisY2AutomaticDivision;
//PE_ADM_* or PE_UNCHANED for no change
    short seriesAxisAutomaticDivision;
//PE_ADM_* or PE_UNCHANED for no change
    long dataAxisYManualDivision;
//if dataAxisYAutomaticDivision is PE_ADM_AUTOMATIC, this field is
ignored
    long dataAxisY2ManualDivision;
    long seriesAxisManualDivision;
 //if seriesAxisAutomaticDivision is PE_ADM_AUTOMATIC, this field is
ignored
    short dataAxisYAutoScale;
    short dataAxisY2AutoScale;
    short seriesAxisAutoScale;
}  PEGraphAxisInfo;
```

Members

StructSize	Specifies the size of the PEGraphAxisInfo structure. Initialize this member to PE_SIZEOF_GRAPH_AXIS_INFO.
groupAxisGridLine	Specifies GridLine option. Use one of the PE_GGT_XXX "Chart Gridline Constants" on page 542, or PE_UNCHANGED for no change.
dataAxisYGridLine	Specifies GridLine option. Use one of the PE_GGT_XXX "Chart Gridline Constants" on page 542, or PE_UNCHANGED for no change.
dataAxisY2GridLine	Specifies GridLine option. Use one of the PE_GGT_XXX "Chart Gridline Constants" on page 542, or PE_UNCHANGED for no change.
seriesAxisGridline	Specifies GridLine option. Use one of the PE_GGT_XXX "Chart Gridline Constants" on page 542, or PE_UNCHANGED for no change.
dataAxisYMinValue	Specifies the minimum value for the axis.
dataAxisYMaxValue	Specifies the maximum value for the axis.
dataAxisY2MinValue	Specifies the minimum value for the axis.
dataAxisY2MaxValue	Specifies the maximum value for the axis.
seriesAxisMinValue	Specifies the minimum value for the axis.
seriesAxisMaxValue	Specifies the maximum value for the axis.
dataAxisYNumberFormat	Specifies the format for the display of numeric values on the chart. Use one of the PE_GNF_XXX "Chart Number Format Constants" on page 544, or PE_UNCHANGED for no change.
dataAxisY2NumberFormat	Specifies the format for the display of numeric values on the chart. Use one of the PE_GNF_XXX "Chart Number Format Constants" on page 544, or PE_UNCHANGED for no change.
seriesAxisNumberFormat	Specifies the format for the display of numeric values on the chart. Use one of the PE_GNF_XXX "Chart Number Format Constants" on page 544, or PE_UNCHANGED for no change.
dataAxisYAutoRange	Boolean, or PE_UNCHANGED for no change. If TRUE, the axis will autorange.
dataAxisY2AutoRange	Boolean, or PE_UNCHANGED for no change. If TRUE, the axis will autorange.
seriesAxisAutoRange	Boolean, or PE_UNCHANGED for no change. If TRUE, the axis will autorange.
dataAxisYAutomaticDivision	PE_ADM_AUTOMATIC, PE_ADM_MANUAL, or PE_UNCHANGED for no change.
dataAxisY2AutomaticDivision	PE_ADM_AUTOMATIC, PE_ADM_MANUAL, or PE_UNCHANGED for no change.
seriesAxisAutomaticDivision	PE_ADM_AUTOMATIC, PE_ADM_MANUAL, or PE_UNCHANGED for no change.

dataAxisYManualDivision	If the corresponding axis m_dataAxisYAutomaticDivision is PE_ADM_AUTOMATIC, this field is ignored.
dataAxisY2ManualDivision	If the corresponding axis m_dataAxisY2AutomaticDivision is PE_ADM_AUTOMATIC, this field is ignored.
seriesAxisManualDivision	If the corresponding axis m_seriesAxisAutomaticDivision is PE_ADM_AUTOMATIC, this field is ignored.
dataAxisYAutoScale	Boolean, or PE_UNCHANGED for no change. If TRUE, the axis will autoscale.
dataAxisY2AutoScale	Boolean, or PE_UNCHANGED for no change. If TRUE, the axis will autoscale.
seriesAxisAutoScale	Boolean, or PE_UNCHANGED for no change. If TRUE, the axis will autoscale.

VB Type Listing

```
Type PEGraphAxisInfo
    StructSize As Integer
    groupAxisGridLine As Integer
    dataAxisYGridLine As Integer
    dataAxisY2GridLine As Integer
    seriesAxisGridline As Integer
    dataAxisYMinValue As Double
    dataAxisYMaxValue As Double
    dataAxisY2MinValue As Double
    dataAxisY2MaxValue As Double
    seriesAxisMinValue As Double
    seriesAxisMaxValue As Double
    dataAxisYNumberFormat As Integer
    dataAxisY2NumberFormat As Integer
    seriesAxisNumberFormat As Integer
    dataAxisYAutoRange As Integer
    dataAxisY2AutoRange As Integer
    seriesAxisAutoRange As Integer
    dataAxisYAutomaticDivision As Integer
    dataAxisY2AutomaticDivision As Integer
    seriesAxisAutomaticDivision As Integer
    dataAxisYManualDivision As Long
    dataAxisY2ManualDivision As Long
    seriesAxisManualDivision As Long
    dataAxisYAutoScale As Integer
    dataAxisY2AutoScale As Integer
    seriesAxisAutoScale As Integer
End Type
```

Delphi Record Listing

```
type
  PEGraphAxisInfo = record
    StructSize         : Word;
    groupAxisGridLine  : Smallint;
    dataAxisYGridLine  : Smallint;
    dataAxisY2GridLine : Smallint;
    seriesAxisGridline : Smallint;
    dataAxisYMinValue  : double;
    dataAxisYMaxValue  : double;
    dataAxisY2MinValue : double;
    dataAxisY2MaxValue : double;
    seriesAxisMinValue : double;
    seriesAxisMaxValue : double;
    dataAxisYNumberFormat  : Smallint;
    dataAxisY2NumberFormat : Smallint;
    seriesAxisNumberFormat : Smallint;
    dataAxisYAutoRange  : Smallint;
    dataAxisY2AutoRange : Smallint;
    seriesAxisAutoRange : Smallint;
    dataAxisYAutomaticDivision  : Smallint;
    dataAxisY2AutomaticDivision : Smallint;
    seriesAxisAutomaticDivision : Smallint;
    dataAxisYManualDivision  : Longint;
    dataAxisY2ManualDivision : Longint;
    seriesAxisManualDivision : LongInt;
  end;
```

PEGraphOptionInfo

PEGraphOptionInfo contains information about chart appearance and is used by “PEGetGraphOptionInfo” on page 311, and “PESetGraphOptionInfo” on page 409.

C Syntax

```
typedef struct PEGraphOptionInfo {
    WORD StructSize;
    short graphColour;
    short showLegend;
    short legendPosition;
    short pieSize;
    short detachedPieSlice;
    short barSize;

    short verticalBars;
    short markerSize;
    short markerShape;
    short dataPoints;
    short dataValueNumberFormat;
    short viewingAngle;
    short legendLayout;
} PEGraphOptionInfo;
```

Members

StructSize	Specifies the size of the PEGraphOptionsInfo structure. Initialize this member to PE_SIZEOF_GRAPH_OPTION_INFO.
graphColour	Use one of the PE_GCR_XXX "Chart Options Constants" on page 541, or PE_UNCHANGED for no change.
showLegend	BOOLEAN. Use TRUE to show the chart legend, FALSE to hide the chart legend, or PE_UNCHANGED for no change.
legendPosition	Use one of the PE_GLP_XXX "Chart Options Constants" on page 541, or PE_UNCHANGED for no change. Valid only if showLegend is TRUE.
pieSize	For pie charts and doughnut charts, use one of the PE_GPS_XXX "Chart Pie Size Constants" on page 544, or PE_UNCHANGED for no change.
detachedPie Slice	For pie charts and doughnut charts, use one of the PE_GDPS_XXX "Chart Slice Detachment Constants" on page 544, or PE_UNCHANGED for no change.
barSize	For bar charts, use one of the PE_GBS_XXX "Chart Bar Size Constants" on page 542, or PE_UNCHANGED for no change.
verticalBars	BOOLEAN. For bar charts, use TRUE if the chart will have vertical bars, FALSE if the chart will not have vertical bars, or PE_UNCHANGED for no change.
markerSize	For line charts and bar charts, use one of the PE_GMS_XXX "Chart Marker Size Constants" on page 543, or PE_UNCHANGED for no change.
markerShape	For line charts and bar charts, use one of the PE_GMSP_XXX "Chart Marker Shape Constants" on page 543, or PE_UNCHANGED for no change.
dataPoints	Use one of the PE_GDP_XXX "Chart Data Point Constants" on page 542, or PE_UNCHANGED for no change.
dataValue NumberFormat	Use one of the PE_GNF_XXX "Chart Number Format Constants" on page 544, or PE_UNCHANGED for no change.
viewingAngle	For 3D charts, use one of the PE_GVA_XXX "Chart Viewing Angle Constants" on page 545, or PE_UNCHANGED for no change.
legendLayout	Specifies the legend layout. Use one of the PE_GLL_XXX "Chart Legend Layout Constants".

VB Type Listing

```
Type PEGraphOptionInfo
    StructSize As Integer
    graphColour As Integer
    showLegend As Integer
    legendPosition As Integer
    pieSize As Integer
    detachedPieSlice As Integer
    barSize As Integer
    verticalBars As Integer
    markerSize As Integer
    markerShape As Integer
    dataPoints  As Integer
    dataValueNumberFormat As Integer
    viewingAngle As Integer
    legendLayout As Integer
End Type
```

Delphi Record Listing

```
type
  PEGraphOptionInfo = record
    StructSize       : Word;
    graphColour      : Smallint;
    showLegend       : Smallint;
    legendPosition   : Smallint;
    pieSize          : Smallint;
    detachedPieSlice : Smallint;
    barSize          : Smallint;
    verticalBars     : Smallint;
    markerSize       : Smallint;
    markerShape      : Smallint;
    dataPoints       : Smallint;
    dataValueNumberFormat : Smallint;
    viewingAngle     : Smallint;
  end
```

PEGraphTypeInfo

PEGraphTypeInfo contains information about the type of the specified chart and is used by PEGetGraphTypeInfo and PESetGraphTypeInfo.

C Syntax

```
typedef struct PEGraphTypeInfo {
    WORD StructSize;
    short graphType;
    short graphSubtype;
} PEGraphTypeInfo;
```

Members

StructSize	Specifies the size of the PEGraphTypeInfo structure. Initialize this member to PE_SIZEOF_GRAPH_TYPE_INFO.
graphType	Uses one of the PE_GT_XXX “Graph Type Constants” on page 555, or PE_UNCHANGED for no change.
graphSubtype	Uses one of the PE_GST_XXX “Graph Subtype Constants” on page 551, or PE_UNCHANGED for no change.

VB Type Listing

```
Type PEGraphTypeInfo
    StructSize As Integer
    graphType As Integer
    graphSubtype As Integer
End Type
```

Delphi Record Listing

```
type
  PEGraphTypeInfo = record
    StructSize   : Word;
    graphType    : Smallint;
    graphSubtype : Smallint;
  end;
```

PEGroupOptions

PEGroupOptions contains information about report group options. This information is used by "PEGetGroupOptions" on page 316 to retrieve current options and by "PESetGroupOptions" on page 414 to pass new options.

C Syntax

```
typedef struct PEGroupOptions {
    WORD StructSize;
    short condition;
    char  fieldName [PE_FIELD_NAME_LEN];
    short sortDirection;
    short repeatGroupHeader;
    short keepGroupTogether;
    short topOrBottomNGroups;
    char  topOrBottomNSortFieldName [PE_FIELD_NAME_LEN];
    short nTopOrBottomGroups;
    short discardOtherGroups;
    short hierarchicalSorting;
    char  instanceIDField [PE_FIELD_NAME_LEN];
    char  parentIDField [PE_FIELD_NAME_LEN];
    long  groupIndent;
} PEGroupOptions;
```

Members

StructSize	Specifies the size of the PEGroupOptions structure. Initialize to PE_SIZEOF_GROUP_OPTIONS.
condition	Specifies the condition setting for the selected group section. When getting, use PE_GC_TYPEMASK and PE_GC_CONDITIONMASK to decode the condition. When setting, pass a PE_GC_XXX "Group Condition Constants" on page 556, or PE_UNCHANGED for no change.
fieldName	Specifies the field name of the group field (of length PE_FIELD_NAME_LEN = 512). Use formula form or leave empty for no change.
sortDirection	Specifies one of the "Sort Order Constants" on page 560 or PE_UNCHANGED for no change.
repeatGroupHeader	TRUE, FALSE, or PE_UNCHANGED for no change.
keepGoupTogether	TRUE, FALSE, or PE_UNCHANGED for no change.

<table>
<tr><td>topOrBottomNGroups</td><td colspan="2">Use one of the following PE_GO_TBN_XXX constants or PE_UNCHANGED for no change.</td></tr>
<tr><td></td><td>Constant</td><td>Description</td></tr>
<tr><td></td><td>PE_GO_TBN_ALL_GROUPS_ UNSORTED</td><td>There is no group sorting or Top / Bottom N for this level of grouping.</td></tr>
<tr><td></td><td>PE_GO_TBN_ALL_GROUPS_ SORTED</td><td>There is group sorting, but not Top / Bottom N.</td></tr>
<tr><td></td><td>PE_GO_TBN_TOP_N_GROUPS</td><td>Top N groups will be selected.</td></tr>
<tr><td></td><td>PE_GO_TBN_BOTTOM_N_GRO UPS</td><td>Bottom N groups will be selected.</td></tr>
<tr><td>topOrBottomNSort FieldName</td><td colspan="2">Specifies the field name (of length PE_FIELD_NAME_LEN = 512) of the summary field by which the groups are ordered. Use formula form or leave empty for no change.</td></tr>
<tr><td>nTopOrBottomGroup s</td><td colspan="2">Specifies the number of groups to keep. Use 0 to keep all groups and PE_UNCHANGED for no change.</td></tr>
<tr><td>discardOtherGroups</td><td colspan="2">Determines whether the remaining groups are collected into an Others group or discarded. TRUE, FALSE, or PE_UNCHANGED for no change.</td></tr>
<tr><td>hierarchicalSorting</td><td colspan="2">Specifies whether or not sorting is hierarchical. TRUE, FALSE, or PE_UNCHANGED for no change.</td></tr>
<tr><td>instanceIDField</td><td colspan="2">Specifies the instance ID field (of length PE_FIELD_NAME_LEN = 512) for hierarchical sorting</td></tr>
<tr><td>parentIDField</td><td colspan="2">Specifies the parent ID field (of length PE_FIELD_NAME_LEN = 512) for hierarchical sorting</td></tr>
<tr><td>groupIndent</td><td colspan="2">Specifies the indent for hierarchical group sorting, in twips.</td></tr>
</table>

Remarks

If topOrBottomNGroups is PE_GO_TBN_TOP_N_GROUPS or PE_GO_TBN_BOTTOM_N_GROUPS, all the group sort fields related to this group will be deleted. A new group sort field will be added with the sort direction of descending or ascending. The group sort field will be sorted by specifying topOrBottomNSortFieldName if it is not empty. It will be sorted by the first group sort field name related to this group (before it is deleted) if topOrBottomNSortFieldName is empty.

VB Type Listing

```
Type PEGroupOptions
    StructSize As Integer
    condition As Integer
    fieldName As String * PE_FIELD_NAME_LEN
    sortDirection As Integer
    repeatGroupHeader As Integer
    keepGroupTogether As Integer
    topOrBottomNGroups As Integer * PE_FIELD_NAME_LEN
    topOrBottomNSortFieldName As String * PE_FIELD_NAME_LEN
    nTopOrBottomGroups As Integer
    discardOtherGroups As Integer
    hierarchicalSorting As Integer
    instanceIDField As String * PE_FIELD_NAME_LEN
    parentIDField As String * PE_FIELD_NAME_LEN
    groupIndent As Long
End Type
```

Delphi Record Listing

```
type
    PEFieldNameType = array[0..PE_FIELD_NAME_LEN-1] of char;
    PEGroupOptions = record
        StructSize: Word;
        condition: smallint;
        fieldName: PEFieldNameType;
        sortDirection: smallint;
        repeatGroupHeader: smallint;
        keepgroupTogether: smallint;
        topOrBottomNGroups: smallint;
        topOrBottomNSortFieldName: PEFieldNameType;
        nTopOrBottomGroups: smallint;
        discardOtherGroups: smallint
    end;
```

PEGroupTreeButtonClickedEventInfo

Provides information about the group tree button clicked event, when the callback function is called with the event ID equal to PE_GROUP_TREE_BUTTON_CLICKED_EVENT.

C Syntax

```
typedef struct PEGroupTreeButtonClickedEventInfo {
    WORD StructSize;
    short visible;
    long windowHandle;
} PEGroupTreeButtonClickedEventInfo;
```

Members

StructSize	Specifies the size of the PEGroupTreeButtonClickedEventInfo structure. Initialize this member to PE_SIZEOF_GROUP_TREE_BUTTON_CLICKED_EVENT_INFO.
visible	Indicates whether the group tree is shown or hidden.
window Handle	Frame window handle where the event happens.

VB Type Listing

```
Type PEGroupTreeButtonClickedEventInfo
    StructSize As Integer
    visible As Integer
    windowHandle As Long
End Type
```

Delphi Record Listing

```
type
    PEGroupTreeButtonClickedEventInfo = record
        StructSize: Word;
        visible: smallint;
        windowHandle: HWnd;
    end;
```

PEHyperlinkEventInfo

PEHyperlinkEventInfo contains information related to the specified hyperlink.

C Syntax

```
typedef struct PEHyperlinkEventInfo {
    WORD StructSize;
    WORD ignored;
    long windowHandle;
    char FAR *hyperlinkText;
} PEHyperlinkEventInfo;
```

Members

StructSize	Specifies the size of the PEHyperlinkEventInfo structure. Initialize this member to PE_SIZEOF_HYPERLINKEVENTINFO.
ignored	For 4-byte alignment. Ignored.
windowHandle	Specifies HWND for the window in which the event occurred.
hyperlinkText	Specifies a pointer to the hyperlink text associated with the specified object. The memory referenced by the pointer is freed after calling the callback function.

PEJobInfo

Contains print job process and display information that is used by "PEGetJobStatus" on page 319.

C Syntax

```
typedef struct PEJobInfo {
    WORD StructSize;
    DWORD NumRecordsRead;
    DWORD NumRecordsSelected;
    DWORD NumRecordsPrinted;
    WORD DisplayPageN;
    WORD LatestPageN;
    WORD StartPageN;
    BOOL printEnded;
} PEJobInfo;
```

Members

StructSize	Specifies the size of the PEJobInfo structure. Initialize to PE_SIZEOF_JOB_INFO.
NumRecordsRead	Specifies the number of records actually processed.
NumRecordsSelected	Specifies the number of records selected for inclusion in the report out of the total number of records read.
NumRecordsPrinted	Specifies the number of records actually printed or previewed.
DisplayPageN	Specifies the page number of the currently displayed page in the preview window.
LatestPageN	Specifies the page being generated. Once the printing is complete, this value is the number of the last page.
StartPageN	Specifies the number of the starting page. The value will normally be 1, but you can specify something else using "PESetPrintOptions" on page 435.
printEnded	Specifies whether or not the printing process is completed. TRUE indicates that this process is completed; FALSE indicates that is not yet complete. When printing to a preview window, printEnded is True only when the last page is reached.

VB Type Listing

```
Type PEJobInfo
    StructSize As Integer
    NumRecordsRead As Long
    NumRecordsSelected As Long
    NumRecordsPrinted As Long
    DisplayPageN As Integer
    LatestPageN As Integer
    StartPageN As Integer
    PrintEnded As Long
Type PEJobInfo
```

Delphi Record Listing

```
type
    PEJobInfo = record
        StructSize: Word;
        NumRecordsSelected: longint;
        NumRecordsPrinted: longint;
        DisplayPageN: Word;
        LatestPageN: Word;
        StartPageN: Word;
        PrintEnded: Bool;
    end;
```

PELaunchSeagateAnalysisEventInfo

PELaunchSeagateAnalysisEventInfo contains information related to the launching of Seagate Analysis.

Note: Seagate Analysis is now known as Crystal Analysis.

C Syntax

```
typedef struct PELaunchSeagateAnalysisEventInfo {
    WORD StructSize;
    WORD ignored;
    long windowHandle;
    char FAR *pathFile;
} PELaunchSeagateAnalysisEventInfo;
```

Members

StructSize	Specifies the size of the PELaunchSeagateAnalysisEventInfo structure. Initialize this member to PE_SIZEOF_LAUNCH_SEAGATE_ANALYSIS_EVENT_INFO.
ignored	For 4-byte alignment. Ignored.
window Handle	Specifies HWND for the window in which the event occurred.
pathFile	Specifies a pointer to the path and filename of the temporary report. The memory referenced by the pointer is freed after calling the callback function.

PELogOnInfo

PELogOnInfo contains log on information that is used by “PEGetNthTableLogOnInfo” on page 347; “PESetNthTableLogOnInfo” on page 427; “PELogOnServer” on page 374; and “PELogOffServer” on page 373 for logging on and off SQL and password-protected non-SQL databases.

C Syntax

```
typedef struct PELogOnInfo {
    WORD StructSize;
    char ServerName [PE_SERVERNAME_LEN];
    char DatabaseName [PE_DATABASENAME_LEN];
    char UserID [PE_USERID_LEN];
    char Password [PE_PASSWORD_LEN];
} PELogOnInfo;
```

Members

StructSize	Specifies the size of the PELogOnInfo structure. Initialize this member to PE_SIZEOF_LOGON_INFO.
ServerName	Specifies the logon name for the server (of length PE_SERVERNAME_LEN = 128, NULL-terminated) used to create the report. See Remarks below.
DatabaseName	Specifies the database logon name (of length PE_DATABASENAME_LEN = 128, NULL-terminated) for the database used to create the report. See Remarks below.
UserID	Specifies the user I.D (of length PE_USERID_LEN = 128, NULL-terminated) necessary to log on to the server. See Remarks below.
Password	Specifies the password (of length PE_PASSWORD_LEN = 128, NULL-terminated) necessary to log on to the server. See Remarks below.

Remarks

- All strings must be null-terminated.
- Password is undefined when getting information from the report.
- Password must be set using “PEGetNthTableLogOnInfo” on page 347. You can pass an empty string (“”) for *ServerName, DatabaseName, or UserID, and* the program will use 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”) for the other parameters as well.
- For Netware SQL, pass the dictionary path name in member ServerName and data path name in member DatabaseName.
- If your report uses a Microsoft Access database via ODBC, the data source indicated in the ServerName parameter must specify an Access database file. An ODBC data source based on the Access driver with no database specified cannot be used at runtime.
- For Essbase databases, pass the Essbase application and database to the DatabaseName member with a comma between each. For example:

```
Sample,Basic
```

VB Type Listing

```
Type PELogOnInfo
    StructSize As Integer
    ServerName As String * PE_SERVERNAME_LEN
    DatabaseName As String * PE_DATABASENAME_LEN
    UserID As String * PE_USERID_LEN
    Password As String * PE_PASSWORD_LEN
End Type
```

Delphi Record Listing

```
type
    PELogonServerType = array[0..PE_SERVERNAME_LEN-1] of char;
    PELogonDBType = array[0..PE_DATABASENAME_LEN-1] of char;
    PELogonUserType = array[0..PE_USERID_LEN-1] of char;
    PELogonPassType = array[0..PE_PASSWORD_LEN-1] of char;
    PELogOnInfo = record
        StructSize: Word;
        ServerName: PELogonServerType;
        DatabaseName: PELogonDbType;
        UserId: PELogonUserType;
        Password: PELogonPassType;
    end;
```

PEMouseClickEventInfo

PEMouseClickEventInfo contains information associated with a mouse click event when the callback function is called with event ID equal to PE_RIGHT/MIDDLE/LEFT_CLICK_EVENT.

C Syntax

```
typedef struct PEMouseClickEventInfo {
    WORD StructSize;
    long windowHandle;
    UINT clickAction;
    UINT clickFlags;
    int xOffset;
    int yOffset;
    PEValueInfo fieldValue;
    DWORD objectHandle;
    short sectionCode
} PEMouseClickedEventInfo;
```

Members

StructSize	Specifies the size of the PEMouseClickEventInfo structure. Initialize this member to PE_SIZEOF_MOUSE_CLICK_EVENT_INFO.		
windowHandle	Specifies the handle of the frame window in which the mouse click event occurred.		
clickAction	Indicates the click action. Uses one of the following PE_MOUSE_XXX constants.		
	Constant	**Description**	
	PE_MOUSE_NOTSUPPORTED		
	PE_MOUSE_DOWN		
	PE_MOUSE_UP		
	PE_MOUSE_DOUBLE_CLICK	For Mouse Left click or Mouse Middle click.	
clickFlags	Indicates the source of the event, which can be any combination of the following PE_CF_XXX virtual key state-mask constants.		
	Constant	**Value**	**Description**
	PE_CF_NONE	0x0000	No button pressed
	PE_CF_LBUTTON	0x0001	Left mouse button
	PE_CF_RBUTTON	0x0002	Right mouse button
	PE_CF_SHIFTKEY	0x0004	Shift key
	PE_CF_CONTROLKEY	0x0008	Control key
	PE_CF_MBUTTON	0x00010	Center mouse button
xOffset	X-coordinate of cursor during mouse click in pixels.		
yOffset	Y-coordinate of cursor during mouse click in pixels.		
fieldValue	The “PEValueInfo” on page 516, structure containing information about the value of the object at the click point, if it is a field object (excluding MEMO and BLOB fields), else valueType element = PE_VI_NOVALUE.		
objectHandle	Specifies the handle of the design view object.		
sectionCode	Specifies the “Section Codes” on page 559 for the section in which the click occurred. See “Working with section codes” on page 46.		

Delphi Record Listing

```
type
   PEMouseClickEventInfo = record
    StructSize : Word;
    windowHandle : LongInt;
    clickAction : integer;
    clickFlags : integer;{
    xOffset : integer;
    yOffset : integer;
    fieldValue : PEValueInfo;
    objectHandle : DWord;
    sectionCode : smallint;
end;
```

PEObjectInfo

PEObjectInfo contains information related to the type, location, and appearance of an object.

C Syntax

```
typedef struct PEObjectInfo {
    WORD StructSize;
    WORD objectType;
    long xOffset;
    long yOffset;
    long width;
    long height;
    short sectionCode;
} PEObjectInfo;
```

Members

StructSize	Specifies the size of the PEObjectInfo structure. Initialize this member to PE_SIZEOF_OBJECT_INFO.
objectType	Specifies the object type. Use one of the PE_OI_XXX “Object Type Constants” on page 558.
xOffset	Specifies the X-offset of the object in the section, in twips, or PE_UNCHANGED for no change.
yOffset	Specifies the Y-offset of the object in the section, in twips, or PE_UNCHANGED for no change.
width	Specifies the width of the object, in twips, or PE_UNCHANGED for no change.
height	Specifies the height of the object, in twips, or PE_UNCHANGED for no change.
sectionCode	Specify the section code for the section containing the object, or PE_UNCHANGED for no change from the current section.

PEParameterFieldInfo

The PEParameterFieldInfo structure contains information related to parameter fields in a report. This structure is used by "PEGetNthParameterField" on page 340 to get information about a specific parameter field and by "PESetNthParameterField" on page 423 to change a specific parameter field.

C Syntax

```
typedef struct PEParameterFieldInfo {
    WORD StructSize;
    WORD ValueType;
    WORD DefaultValueSet;
    WORD CurrentValueSet;
    char Name [PE_PF_NAME_LEN];
    char Prompt [PE_PF_PROMPT_LEN];
    char DefaultValue [PE_PF_VALUE_LEN];
    char CurrentValue [PE_PF_VALUE_LEN];
    char ReportName [PE_PF_REPORT_NAME_LEN];
    WORD needsCurrentValue;
    WORD isLimited;
    double MinSize;
    double MaxSize;
    char EditMask [PE_PF_EDITMASK_LEN];
    WORD isHidden;
} PEParameterFieldInfo;
```

Members

If you wish to set a parameter to NULL, set the CurrentValue to CRWNULL. CRWULL is of Type String and is independent of the data type of the parameter.

StructSize	Specifies the size of the PEParameterFieldInfo structure. Initialize this member to PE_SIZEOF_PARAMETER_FIELD_INFO.
ValueType	Specifies the data type of the parameter field. The Crystal Report Engine supports the data types and associated PE_PF_XXX "Parameter Field Value Type Constants" on page 559.
DefaultValueSet	For backward compatibility for existing applications only. For all new development set this member to FALSE and use PEAdd/GetNth/SetNthParameterDefaultValue. See Remarks below. This member indicates whether a value is assigned to the DefaultValue parameter field. Use TRUE to indicate that a new value is set or FALSE to indicate no change.
CurrentValueSet	For backward compatibility for existing applications only. For all new development set this member to FALSE and use PEAdd/GetNth/SetNthParameterCurrentValue. See Remarks below. This member indicates whether a value is assigned to the CurrentValue parameter field. Use TRUE to indicate that a new value is set or FALSE to indicate no change.
Name	Specifies the name of the parameter field (of length PE_PF_NAME_LEN = 256, NULL-terminated) to be assigned a new value.

Prompt	Specifies the prompting text (of length PE_PF_PROMPT_LEN = 256, NULL-terminated), if any, that will appear when the user runs the report for the first time or refreshes the data.
DefaultValue	For existing applications only; not for new development. See Remarks below. Specifies the default value assigned to the parameter field. If the DefaultValueSet member is FALSE, this value is meaningless. DefaultValue can be a Number, Currency, Date, DateTime, Time, Boolean, or String (of length PE_PF_VALUE_LEN = 256).
CurrentValue	For existing applications only; not for new development. See Remarks below. Specifies the current value assigned to the parameter field. If CurrentValueSet is FALSE, this value is meaningless. CurrentValue can be a Number, Currency, Date, DateTime, Time, Boolean, or String (of length PE_PF_VALUE_LEN = 256).
ReportName	The name of the report (of length PE_PF_REPORT_NAME_LEN = 128) where the field belongs (used only with "PEGetNthParameterField" on page 340).
needsCurrentValue	Returns FALSE if the parameter is linked, not in use, or has current value set.
isLimited	For string values, this will be TRUE if the string is limited on length. For other types, it will be TRUE if the parameter is limited by a range. This capability is not supported currently in Web Viewers.
MinSize	Use for numeric and string fields. Depending on the value type, contains the minimum length of the string or minimum numeric value. For non-numeric and non-string fields (for example, Date/Time), use "PEGetParameterMinMaxValue" on page 351, and "PESetParameterMinMaxValue" on page 431. This capability is not supported currently in Web Viewers.
MaxSize	Use for numeric and string fields. Depending on the value type, contains the maximum length of the string or maximum numeric value. For non-numeric and non-string fields (for example, Date/Time), use "PEGetParameterMinMaxValue" on page 351, and "PESetParameterMinMaxValue" on page 431. This capability is not supported currently in Web Viewers.
EditMask	An edit mask (of length PEP_PF_EDITMASK_LEN = 256) that restricts what may be entered for string parameters. This capability is not supported currently in Web Viewers.
isHidden	TRUE if an essbase sub var. This capability is not supported currently in Web Viewers.

Remarks

- Regarding members DefaultValueSet, CurrentValueSet, DefaultValue, and CurrentValue:
  - To support backward compatibility for existing applications, when Default/CurrentValueSet is set to TRUE, the values in Default/CurrentValue will override any values set by PEAdd/Get/SetParameterDefault/CurrentValue.
  - PEParameterFieldInfo does not process Max or Min parameters if they are passed in after a Date, Time or Date/Time parameter.

- For new application development, the Default/CurrentValueSet must be set to FALSE, so that the Default/CurrentValue members of PEParameterFieldInfo will be ignored and the values associated with PEAdd/Get/SetNthParameterDefault/CurrentValue will be valid. Developers should use the following calls in new applications to access default and current value lists.

  "PEAddParameterCurrentRange" on page 278
  "PEAddParameterCurrentValue" on page 279
  "PEAddParameterDefaultValue" on page 280
  "PEGetNParameterCurrentRanges" on page 325
  "PEGetNParameterCurrentValues" on page 325
  "PEGetNParameterDefaultValues" on page 326
  "PEGetNthParameterCurrentRange" on page 336
  "PEGetNthParameterCurrentValue" on page 337
  "PEGetNthParameterDefaultValue" on page 338
  "PESetNthParameterDefaultValue" on page 422

- To determine if a parameter field is a stored procedure, use "PEGetNthParameterType" on page 341, or "PEGetNthParameterField" on page 340.

VB type listing

```
Type PEParameterFieldInfo
    StructSize As Integer
    ValueType As Integer
    DefaultValueSet As Integer
    CurrentValueSet As Integer
    Name As String * PE_PF_NAME_LEN
    Prompt As String * PE_PF_PROMPT_LEN
    DefaultValue As String * PE_PF_VALUE_LEN
    CurrentValue As String * PE_PF_VALUE_LEN
    ReportName As String * PE_PF_REPORT_NAME_LEN
    needsCurrentValue As Integer
    isLimited As Integer
    MinSize As Double
    MaxSize As Double
    EditMask As String * PE_PF_EDITMASK_LEN
    isHidden As Integer
End Type
```

Delphi listing

```
type
    PEParameterFieldValueType = array[0..PE_PF_NAME_LEN-1] of char;
    PE_PF_ReportNameType = array[0..PE_PF_REPORT_NAME_LEN-1] of char;
    PEParameterFieldNameType  = array [0..PE_PF_NAME_LEN-1] of Char;
    PEParameterFieldEditMaskType = array [0..PE_PF_EDITMASK_LEN-1] of
                                Char;

    PEParameterFieldInfo = record
        structSize: Word;
        ValueType: Word;
        DefaultValueSet: Word;
        CurrentValueSet: Word;
        Name: PEParameterFieldNameType;
        Prompt: PEParameterFieldTextType;
        DefaultValue: PEParameterFieldValueType;
        CurrentValue: PEParameterFieldValueType;
        ReportName: PE_PF_ReportNameType;
        needsCurrentvalue: Word;
        isLimited: Word;
        MinSize: double;
        MaxSize: double;
        EditMask: PEParameterFieldEditMaskType;
        isHidden: Word;
    end;
```

PEParameterPickListOption

PEParameterPickListOption contains information related to parameter sort methods.

C Syntax

```
typedef struct PEParameterPickListOption {
        WORD  StructSize;
        short showDescOnly;
        short sortMethod;
        short sortBasedOnDesc;
} PEParameterPickListOption;
```

Members

StructSize	Specifies the size of the PEParameterFieldInfo structure. Initialize this member to PE_SIZEOF_PICK_LIST_OPTION.
showDescOnly	Boolean, or PE_UNCHANGED for no change.
sortMethod	Use one of thePE_OR_XXX “Sort Method Constants” on page 560, or PE_UNCHANGED for no change.
sortBasedOnDesc	Boolean, or PE_UNCHANGED for no change.

VB Type Listing

```
Type PEParameterPickListOption
    StructSize As Integer
    showDescOnly As Integer
    sortMethod As Integer
    sortBasedOnDesc As Integer
End Type
```

Delphi Record Listing

```
type
  PEParameterPickListOption = record
    StructSize      : Word;
    showDescOnly    : Smallint;
    sortMethod      : Smallint;
    sortBasedOnDesc : Smallint;
  end;
```

PEParameterValueInfo

PEParameterValueInfo contains information about the type of value(s) that a specified parameter field can hold. See “Working with section codes” on page 46.

C Syntax

```
typedef struct PEParameterValueInfo {
    WORD StructSize;
    short isNullable;
    short disallowEditing;
    short allowMultipleValues;
    short hasDiscreteValues;
    short partOfGroup;
    short groupNum;
    short mutuallyExclusiveGroup
} PEParameterValueInfo;
```

Members

StructSize	Specifies the size of the PEParameterValueInfo structure. Set this member to PE_SIZEOF_PARAMETER_VALUE_INFO
isNullable	Specifies whether or not the parameter field can be set to NULL. Set to TRUE, FALSE or PE_UNCHANGED if no change.
disallowEditing	Indicates whether the parameter field value can be edited. Set to TRUE, FALSE, or PE_UNCHANGED if no change.
allowMultipleValues	Specifies whether or not the parameter field can contain multiple values. Set to TRUE, FALSE or PE_UNCHANGED if no change.
hasDiscreteValues	Specifies whether or not the parameter field contains discreet values, range values, or both. Uses one of the following PE_DR_XXX Constants. See “Working with Parameter Values and Ranges” on page 45.

	Constant	Description
	PE_DR_HASRANGE	Only ranges are present.
	PE_DR_HASDISCRETE	Only discrete values are present.
	PE_DR_HASDISCRETEANDRANGE	Both discrete values and ranges are present.
partOfGroup	Specifies whether or not the parameter field is a member of a group. Set to TRUE, FALSE or PE_UNCHANGED if no change.	
groupNum	Specifies the group number or set to PE_UNCHANGED if no change.	
mutuallyExclusive Group	Specifies whether or not the parameter field is a member of a mutually exclusive group. Set to TRUE, FALSE or PE_UNCHANGED if no change.	

VB Type Listing

```
Type PEParameterValueInfo
     StructSize As Integer
     isNullable As Integer
     disallowEditing As Integer
     allowMultipleValues As Integer
     hasDiscreteValues As Integer
     partOfGroup As Integer
     groupNum As Integer
     mutuallyExclusiveGroup As Integer
  End Type
```

Delphi Record Listing

```
type
    PEParameterValueInfo = record
    StructSize : Word;
   isNullable : smallint;
   disallowEditing : smallint;
   allowMultipleValues : smallint;
   hasDiscreteValues : smallint;
   partOfGroup : smallint;{
   groupNum : smallint;
   mutuallyExclusiveGroup : smallint;
end;
```

PEPrintOptions

PEPrintOptions contains printing specifications that are used by the “PEGetPrintOptions” on page 356, to retrieve current options and “PESetPrintOptions” on page 435, to pass new options. These specifications are the same as those that can be set using the Print common dialog box.

C Syntax

```
typedef struct PEPrintOptions {
    WORD StructSize;
    unsigned short startPageN;
    unsigned short stopPageN;
    unsigned short nReportCopies;
    unsigned short collation;
    char outputFileName [PE_FILE_PATH_LEN];
} PEPrintOptions;
```

Members

<table>
<tr><td>StructSize</td><td colspan="2">Specifies the size of the PEPrintOptions structure. Initialize this member to PE_SIZEOF_PRINT_OPTIONS.</td></tr>
<tr><td>startPageN</td><td colspan="2">Specifies the first page that you want to print. Page numbers are 1-based (Page 1 = 1, Page 2 = 2, etc.). Use 0 if you want to retain the existing settings.</td></tr>
<tr><td>stopPageN</td><td colspan="2">Specifies the last page that you want to print. Page numbers are 1-based (Page 1 = 1, Page 2 = 2, etc.). Use 0 if you want to retain the existing settings.</td></tr>
<tr><td>nReportCopies</td><td colspan="2">Specifies the number of copies that you want to print. Copy numbers, like page numbers, are 1-based. Use 0 if you want to retain the existing settings.</td></tr>
<tr><td>collation</td><td colspan="2">Indicates whether or not you want the program to collate the copies (if you are printing multiple copies of a multiple page report). For this parameter, use one of the following constants:</td></tr>
<tr><td></td><td>Constant</td><td>Description</td></tr>
<tr><td></td><td>PE_UNCOLLATED</td><td>Prints multiple copies of a multiple page report uncollated (Page order = 1, 1, 1, 2, 2, 2, 3, 3, 3, etc.).</td></tr>
<tr><td></td><td>PE_COLLATED</td><td>Prints multiple copies of a multiple page report collated (Page order = 1, 2, 3,..., 1, 2, 3,..., etc.).</td></tr>
<tr><td></td><td>PE_DEFAULTCOLLATION</td><td>Prints multiple copies of a multiple page report using the collation settings as specified in the report.</td></tr>
<tr><td>outputFileName</td><td colspan="2">Specifies a path and file name (of length PE_FILE_PATH_LEN = 512) if the report is printed to a file.</td></tr>
</table>

VB Type Listing

```
Type PEPrintOptions
    StructSize As Integer
    StartPageN As Integer
    StopPageN As Integer
    nReportCopies As Integer
    collation As Integer
    outputFileName As String * PE_FILE_PATH_LEN
End Type
```

Delphi Record Listing

```
type
  PEOutputFileNameType = array [0..PE_FILE_PATH_LEN-1] of Char;
  PEPrintOptions  = record
        StructSize: Word;
        StartPageN: Word;
        StopPageN: Word;
        nReportCopies: Word;
        Collation: Word;
        outputFileName: PEOutputFileNameType ;
    end;
```

PEReadingRecordsEventInfo

PEReadingRecordsEventInfo contains information about records read when a callback function is called with event ID equal to PE_READING_RECORDS_EVENT.

C Syntax

```
typedef struct PEReadingRecordsEventInfo {
    WORD StructSize;
    short cancelled;
    long recordsRead;
    long recordsSelected;
    short done;
} PEReadingRecordsEventInfo;
```

Members

StructSize	Specifies the size of the PEReadingRecordsEventInfo structure. Initialize this member to PE_SIZEOF_READING_RECORDS_EVENT_INFO.
cancelled	Boolean value indicates whether the reading records is canceled.
recordsRead	Indicates how many records have been read.
recordsSelected	Indicates how many records have been selected.
done	Boolean value indicates whether reading records is finished.

VB Type Listing

```
Type PEReadingRecordsEventInfo
    StructSize As Integer
    cancelled As Integer
    recordsRead As Long
    recordsSelected As Long
    done As Integer
End Type
```

Delphi Record Listing

```
type
    PEReadingRecordsEventInfo = record
        StructSize: Word;
        cancelled: smallint;
        recordsRead: longint;
        recordsSelected: longint;
        done: smallint;
end;
```

PEReportFieldMappingInfo

PEReportFieldMappingInfo contains information required to associate (map) a report field to a database field that has been changed.

C Syntax

```
typedef struct PEReportFieldMappingInfo {
    WORD StructSize;
    WORD valueType;
    char tableAliasName[PE_TABLE_NAME_LEN];
    char databaseFieldName[PE_DATABASE_FIELD_NAME_LEN];
    int mappingTo;
} PEReportFieldMappingInfo;
```

Members

<table>
<tr><td>StructSize</td><td colspan="2">Specifies the size of the PEReportFieldMappingInfo structure. Initialize this member to PE_SIZEOF_REPORT_FIELDMAPPING_INFO.</td></tr>
<tr><td>valueType</td><td colspan="2">Indicates the field value type. The Value Type can be one of the following constants:</td></tr>
<tr><td></td><td>Constant</td><td>Description</td></tr>
<tr><td></td><td>PE_FVT_INT8SFIELD</td><td>8-bit integer signed</td></tr>
<tr><td></td><td>PE_FVT_INT8UFIELD</td><td>8-bit integer unsigned</td></tr>
<tr><td></td><td>PE_FVT_INT16SFIELD</td><td>16-bit integer signed</td></tr>
<tr><td></td><td>PE_FVT_INT16UFIELD</td><td>16-bit integer unsigned</td></tr>
<tr><td></td><td>PE_FVT_INT32SFIELD</td><td>32-bit integer signed</td></tr>
<tr><td></td><td>PE_FVT_INT32UFIELD</td><td>32-bit integer unsigned</td></tr>
<tr><td></td><td>PE_FVT_NUMBERFIELD</td><td>Number field</td></tr>
</table>

	PE_FVT_CURRENCYFIELD	Currency field
	PE_FVT_BOOLEANFIELD	Boolean field
	PE_FVT_DATEFIELD	Date field
	PE_FVT_TIMEFIELD	Time field
	PE_FVT_STRINGFIELD	String field
	PE_FVT_TRANSIENTMEMOFIELD	Transient Memo field
	PE_FVT_PERSISTENTMEMOFIELD	Persistent Memo field
	PE_FVT_BLOBFIELD	BLOB field
	PE_FVT_DATETIMEFIELD	Date / Time field
	PE_FVT_BITMAPFIELD	Bitwise field
	PE_FVT_ICONFIELD	Icon field
	PE_FVT_PICTUREFIELD	Picture field
	PE_FVT_OLEFIELD	OLE field
	PE_FVT_GRAPHFIELD	Graph field
	PE_FVT_UNKNOWNFIELD	Unknown field type
tableAliasName	A string (of length PE_TABLE_NAME_LEN = 128) which contains the database table alias name.	
databaseFieldName	A string of length (of length PE_DATABASE_FIELD_NAME_LEN = 128) which contains the name of the database field.	
mappingTo	An index of a field in an array in the databaseField member of “PEFieldMappingEventInfo” on page 462, which contains the list of database fields. If the field is unmapped then this value is -1.	

Delphi Record Listing

```
type
  PETableAliasNameType = Array [0..PE_TABLE_NAME_LEN -1] of Char;
  PEDatabaseFieldNameType = Array [0..PE_DATABASE_FIELD_NAME_LEN -1]
                                  of Char;
  PEReportFieldMappingInfo = record
        StructSize : Word;
        valueType : Word;
        tableAliasName : PETableAliasNameType;
        databaseFieldName : PEDatabaseFieldNameType;
        mappingTo : integer;
        PEFieldMappingEventInfo->databaseFields}
end;
```

PEReportAlertInfo

PEReportAlertInfo contains information on Report Alerts. This information is used by "PEGetNthReportAlert" on page 343, to retrieve information on the Report Alerts in the report.

C Syntax

```
typedef struct PEReportAlertInfo
{
    WORD StructSize;
    short nameLength;
    HANDLE name;
    short isEnabled;
    short alertConditionLength;
    HANDLE alertConditionFormula;
    DWORD nTriggeredInstances;
    short alertMessageLength;
    short defaultAlertMessageLength;
    HANDLE alertMessageFormula;
    HANDLE defaultAlertMessage;
} PEReportAlertInfo;
```

Members

Release the HANDLE even if you are not using it.

StructSize	Specifies the size of the PEReportAlertInfo structure. Initialize this member to PE_SIZEOF_REPORT_ALERT_INFO.
namelenght	Specifies the length of the Report Alert's name.
name	Specifies a handle to the string containing the name of the Report Alert. This is the name assigned to the Report Alert when it was first created.
isEnabled	Specifies wether or not the Report Alert is enabled. Pass TRUE to enable, or PE_UNCHANGED for no change.
alertCondition Lenght	Specifies the length of the condition defined for the Report Alert.
alertCondition Formula	Specifies a handle to the condition formula of the Report Alert. This is the formula created to trigger the Report Alert.
nTriggered Instances	Specifies the number of times a Report Alert was triggered.
alertMessage Lenght	Specifies the length of the conditional message defined for the Report Alert.
defaultAlert MessageLenght	Specifies the length of the default message defined for the Report Alert.
alertMessage Formula	Specifies a handle to the formula used to create the alert message when a Report Alert is triggered.
defaultAlertMessge	Specifies a handle to the default Report Alert message.

VB Type Listing

```
Type PEReportAlertInfo
    StructSize As Integer
    nameLength As Integer
    name as Long
    isEnabled As Integer
    alertConditionLenght As Integer
    alertConditionFormula As Long
    nTriggeredInstances As Long
    alertMessageLength As Integer
    defaultAlertmessageLength As Integer
    alertMessageFormula As Long
    defaultAlertMessage as Long
End Type
```

Delphi Record Listing

```
type PEReportAlertInfo = record
    StructSize : Word;
    nameLength : Smallint;
    name : HWND;
    isEnabled : Smallint;
    alertConditionLength : Smallint;
    alertConditionFormula : HWND;
    nTriggeredInstances : DWord;
    alertMessageLength : SmallInt;
    defaultAlertMessageLength : SmallInt;
    alertMessageFormula : HWND;
    defaultAlertMessage : HWND;
end;
```

PEReportOptions

PEReportOptions contains report option information. This information is used by "PEGetReportOptions" on page 357, to retrieve current options and by "PESetReportOptions" on page 436, to pass new options.

C Syntax

```
typedef struct PEReportOptions {
    WORD StructSize;
    short saveDataWithReport;
    short saveSummariesWithReport;
    short useIndexForSpeed;
    short translateDOSStrings;
    short translateDOSMemos;
    short convertDateTimeType;
    short convertNullFieldToDefault;
    short morePrintEngineErrorMessages;
    short caseInsensitiveSQLData;
    short verifyOnEveryPrint;
    short zoomMode;
    short hasGroupTree;
    short dontGenerateDataForHiddenObjects;
    short performGroupingOnServer;
    short doAsyncQuery;
    short promptMode;
    short selectDistinctRecords;
    short alwaysSortLocally;
    short isReadOnly;
    short canSelectDistinctRecords;
}PEReportOptions;
```

Members

For each member, except as noted below, use TRUE, FALSE, or PE_UNCHANGED for no change.

structSize	Specifies the size of the PEReportOptions structure. Initialize to PE_SIZEOF_REPORT_OPTIONS.
saveDataWithReport	Specifies whether or not data should be saved with the report. Boolean, or PE_UNCHANGED for no change.
saveSummariesWithReport	Specifies whether to save summaries with the report. Boolean, or PE_UNCHANGED for no change.
useIndexForSpeed	Specifies whether or not to use index values. Boolean, or PE_UNCHANGED for no change.
translateDOSStrings	Specifies whether or not to translate DOS strings. Boolean, or PE_UNCHANGED for no change.

translateDOSMemos	Specifies whether or not to translate DOS memos. Boolean, or PE_UNCHANGED for no change.		
convertDateTimeType	Specifies whether or not to convert DATE/Time format to another format. Use one of the following constants, or PE_UNCHANGED for no change.		
	Constant	**Value**	
	PE_RPTOPT_CVTDATETIMETOSTR	0	
	PE_RPTOPT_CVTDATETIMETODATE	1	
	PE_RPTOPT_KEEPDATETIMETYPE	2	
convertNullFieldToDefault	Specifies whether or not to convert NULL parameter fields to their default values. Boolean, or PE_UNCHANGED for no change.		
morePrintEngineError Messages	Specifies whether or not to allow the print engine to pop up error messages to the screen from an application without the users intervention. Boolean, or PE_UNCHANGED for no change.		
caseInsensitiveSQLData	Specifies whether or not to perform a case insensitive search for SQL data. Boolean, or PE_UNCHANGED for no change.		
verifyOnEveryPrint	Specifies whether or not to perform database verification for every print job. Boolean, or PE_UNCHANGED for no change.		
zoomMode	Use one of the “Zoom Level Constants” on page 561, or PE_UNCHANGED for no change.		
hasGroupTree	Specifies whether or not there is a group tree associated with the report. Boolean, or PE_UNCHANGED for no change.		
dontGenerateData ForHiddenObjects	Specifies whether or not to generate data for hidden objects. Boolean, or PE_UNCHANGED for no change.		
performGroupingOnServe r	Specifies whether or not to perform grouping on servers. Boolean, or PE_UNCHANGED for no change.		
doAsyncQuery	Boolean, or PE_UNCHANGED for no change.		
promptMode	Specifies the prompt mode. Use one of the following constants, or PE_UNCHANGED for no change.		
	Constant	**Value**	**Description**
	PE_RPTOPT_PROMPT_NONE	0	
	PE_RPTOPT_PROMPT_NORMAL	1	
	PE_RPTOPT_PROMPT_ALWAYS	2	
selectDistinctRecords	Specifies whether or not to select distinct records. Boolean, or PE_UNCHANGED for no change.		
alwaysSortLocally	Specifies whether or not to sort the records locally. Boolean, or PE_UNCHANGED for no change.		
isReadOnly	Specifies wether the report is read only. Boolean value. This property is read only.		
canSelectDistinctRecords	Specifies wether the report can select distinct records. Boolean value. This property is read only.		

VB Type Listing

```
Type PEReportOptions
    StructSize As Integer
    saveDataWithReport As Integer
    saveSummariesWithReport As Integer
    useIndexForSpeed As Integer
    translateDOSStrings As Integer
    translateDOSMemos As Integer
    convertDateTimeType As Integer
    convertNullFieldToDefault As Integer
    morePrintEngineErrorMessages As Integer
    caseInsensitiveSQLData As Integer
    verifyOnEveryPrint As Integer
    zoomMode As Integer
    hasGroupTree As Integer
    dontGenerateDataForHiddenObjects As Integer
    performGroupingOnServer As Integer
    doAsyncQuery As Integer
    promptMode As Integer
    selectDistinctRecords As Integer
    wysiwygMode As Integer
    alwaysSortLocally as Integer
    isReadOnly as Integer
    canSelectDistinctRecords as Integer
End Type
```

Delphi Record Listing

```
type
    PEReportOptions = record
        StructSize : Word;
        saveDataWithReport : Smallint;
        saveSummariesWithReport : Smallint;
        useIndexForSpeed : Smallint;
        translateDOSStrings : Smallint;
        translateDOSMemos : Smallint;
        convertDateTimeType : Smallint;
        convertNullFieldToDefault : Smallint;
        morePrintEngineErrorMessages : Smallint;
        caseInsensitiveSQLData : Smallint;
        verifyOnEveryPrint : Smallint;
        zoomMode : Smallint;
        hasGroupTree : Smallint;
        dontGenerateDataForHiddenObjects : Smallint;{
        performGroupingOnServer : Smallint;
        alwaysSortLocally : Smallint;
        isReadOnly : Smallint
        canSelectDistinctRecords : Smallint
end;
```

PEReportSummaryInfo

PEReportSummaryInfo contains report summary information, corresponding to the report summary information in the Crystal Reports Designer.

C Syntax

```
typedef struct PEReportSummaryInfo {
    WORD StructSize;
    char applicationName[PE_APPLICATION_NAME_LEN];
    char title[PE_TITLE_LEN];
    char subject[PE_SI_SUBJECT_LEN];
    char author[PE_SI_AUTHOR_LEN];
    char keywords[PE_SI_KEYWORDS_LEN];
    char comments[PE_SI_COMMENTS_LEN];
    char reportTemplate[PE_SI_REPORT_TEMPLATE_LEN];
    short savePreviewPicture;
} PEReportSummaryInfo;
```

Members

StructSize	Specifies the size of the structure. Initialize this member to PE_SIZEOF_REPORT_SUMMARY_INFO.
application Name	Specifies an application name (of length PE_APPLICATION_NAME_LEN = 128) for the application using this report. This member is read only.
title	Specifies the title (of length PE_TITLE_LEN = 128) of the current report.
subject	Specifies the subject (of length PE_SI_SUBJECT_LEN = 128) of the current report.
author	Specifies the (of length PE_SI_AUTHOR_LEN = 128) author of the current report.
keywords	Specifies the keywords (of length PE_SI_KEYWORDS_LEN = 128) included for the current report.
comments	Specifies any comments (of length PE_SI_COMMENTS_LEN = 512) for the current report.
reportTemplate	Specifies the report template (of length PE_SI_REPORT_TEMPLATE_LEN = 128) for the current report.
savePreview Picture	Specifies whether or not to save the preview picture. Boolean or PE_UNCHANGED for no change.

VB Type Listing

```
Type PEReportSummaryInfo
    StructSize As Integer
    applicationName As String * PE_SI_APPLICATION_NAME_LEN' Read only.
    title As String * PE_SI_TITLE_LEN
    subject As String * PE_SI_SUBJECT_LEN
    author As String * PE_SI_AUTHOR_LEN
    keywords As String * PE_SI_KEYWORDS_LEN
    comments As String * PE_SI_COMMENTS_LEN
    reportTemplate As String * PE_SI_REPORT_TEMPLATE_LEN
    savePreviewPicture As Integer
End Type
```

Delphi Record Listing

```
type
    PEApplicationNameType
        = array[0..PE_SI_APPLICATION_NAME_LEN-1] of char;
    PETitleType = array[0..PE_SI_TITLE_LEN-1] of char;
    PESubjectType = array[0..PE_SI_SUBJECT_LEN-1] of char;
    PEAuthorType = array[0..PE_SI_AUTHOR_LEN-1] of char;
    PEKeywordsType = array[0..PE_SI_KEYWORDS_LEN-1] of char;
    PECommentsType = array[0..PE_SI_COMMENTS_LEN-1] of char;
    PEReportTemplate
        = array[0..PE_SI_REPORT_TEMPLATE_LEN-1] of char;

    PEReportSummary = record
        StructSize: Integer;
        applicationName: PEApplicationNameType;
        title: PETitleType;
        subject: PESubjectType;
        author: PEAuthorType;
        keywords: PEKeywordsType;
        comments: PEReportTemplate;
    end;
```

PESearchButtonClickedEventInfo

PESearchButtonClickedEventInfo contains information about a search button clicked event. When the search button is clicked, a callback function will be called with EventId = PE_SEARCH_BUTTON_CLICKED_EVENT.

C Syntax

```
typedef struct PESearchButtonClickedEventInfo {
    long windowHandle;
    char searchString [PE_SEARCH_STRING_LEN];
    WORD StructSize;
} PESearchButtonClickedEventInfo;
```

Members

windowHandle	Specifies the handle of the frame window on which the button is placed.
searchString	Search string (of length PE_SEARCH_STRING_LEN = 128) in search edit control.
StructSize	Specifies the size of the PESearchButtonClickedEventInfo structure. Initialize this member to PE_SIZEOF_SEARCH_BUTTON_CLICKED_EVENT_INFO.

VB Type Listing

```
Type PESearchButtonClickedEventInfo
    windowHandle As Long
    searchString As String * PE_SEARCH_STRING_LEN
    StructSize As Integer
End Type
```

Delphi Record Listing

```
type
    PESearchStringType
    = array[0..PE_SEARCH_STRING_LEN-1] of char;
    PESearchButtonClickedEventInfo = record
        windowHandle: longint;
        searchString: PESearchStringType;
        StructSize: Word;
end;
```

PESectionOptions

PESectionOptions contains specifications for formatting selected report sections and areas. This information is used by the "PEGetSectionFormat" on page 361 and "PEGetAreaFormat" on page 301 to retrieve current settings and "PESetSectionFormat" on page 438 and "PESetAreaFormat" on page 391 to pass new settings.

C Syntax

```
typedef struct PESectionOptions {
    WORDStructSize;
    WORDshort visible;
    WORDnewPageBefore;
    WORDnewPageAfter;
    WORDkeepTogether;
    WORDsuppressBlankSection;
    WORDresetPageNAfter;
    WORDprintAtBottomOfPage;
    COLORREF backgroundColor;
    short underlaySection;
    short showArea;
    short freeFormPlacement;
    short reserveMinimumPageFooter;
} PESectionOptions;
```

Members

StructSize	Specifies the size of the PESectionOptions structure. Initialize to PE_SIZEOF_SECTION_OPTIONS.
visible	Specifies whether or not the selected section is to be visible. Pass TRUE to keep the section visible, FALSE to hide the section, or PE_UNCHANGED for no change.
newPageBefore	Specifies whether or not the program is to insert a page break before the section is printed. Pass TRUE to insert a page break, FALSE to not insert a page break, or PE_UNCHANGED for no change.
newPageAfter	Specifies whether or not the program is to insert a page break after the section is printed. Pass TRUE to insert a page break, FALSE to not insert a page break, or PE_UNCHANGED for no change.

keepTogether	Specifies whether or not the program is to keep the section together, either on the current page (if there is room) or on the next (if not). Pass TRUE to keep the section together, FALSE to allow the program to split the section data from one page to the next if necessary, or PE_UNCHANGED for no change.
suppressBlank Section	Specifies whether or not the program is to eliminate blank sections from your report. A section must be completely empty before the program suppresses it. Pass TRUE to eliminate the blank sections, FALSE to retain them, or PE_UNCHANGED for no change.
resetPageNAfter	Specifies whether or not the program is to reset the page number to one (1) for the following page, after it prints a group total. Pass TRUE to reset the page number, FALSE to retain the standard numbering, or PE_UNCHANGED for no change.
printAtBottomOf Page	Specifies whether or not the program is to cause each group value to print only at the bottom of a page; FALSE to have the values print in their normal position, or PE_UNCHANGED for no change.
backgroundColor	Specifies the RGB color value contained in the “COLORREF” on page 533 value, for section formats only. Use PE_UNCHANGED_COLOR to preserve the existing color or PE_NO_COLOR for no color.
underlaySection	Indicates whether or not the specified section is to underlay the following section(s). TRUE, FALSE, or PE_UNCHANGED for no change.
showArea	Specifies TRUE to show an area, FALSE to hide an area, or PE_UNCHANGED for no change. The user can drill down on a hidden area.
freeFormPlacement	Design time flag. If set to TRUE, an object can be placed anywhere in a section. Use PE_UNCHANGED for no change
reserveMinimum PageFooter	Used to reduce unnecessary white space in the page footer area containing more than one conditionally formatted section. When set to TRUE, the space required to display only one section (the tallest) is reserved. When set to FALSE (default), the maximum height necessary to display every section in the page footer area at full height will be reserved. See Remarks below.

Remarks

- Not all options are available for all sections or areas. For example, if you select a Page Header or Page Footer section or area, you cannot change newPageBefore or newPageAfter. The only options that are available for all sections are visible and suppressBlankSection.
- showArea only applies to area format.
- backgroundColor, underlaySection, showArea, and freeFormPlacement do not apply to area format.
- reserveMinimumPageFooter can be used to remove undesirable white space when there are two or more sections in the page footer area and only one section, based on conditional settings, will be visible when the report is viewed in the preview window. For example, a report page footer area might contain two sections, one conditionally set to display at the bottom of odd pages and the other at the bottom of even pages. When this member is set to TRUE,

sufficient space will be reserved to display only one of the sections included in the area (set to the height of the tallest section). Note that when reserveMinimumPageFooter = TRUE and there is more than one section visible in the page footer area, the visible sections will be displayed only to the extent allowed by the reserved minimum space (the height of the tallest section) with the remainder truncated.

VB Type Listing

```
Type PESectionOptions
    StructSize As Integer
    Visible As Integer
    NewPageBefore As Integer
    NewPageAfter As Integer
    KeepTogether As Integer
    SuppressBlankSection As Integer
    ResetPageNAfter As Integer
    PrintAtBottomOfPage As Integer
    BackgroundColor As Long
    UnderlaySection As Integer
    ShowArea As Integer
    FreeFormPlacement As Integer
    reserveMinimumPageFooter As Integer
End Type
```

Delphi Record Listing

```
type
    PESectionOptions = record
        StructSize: Word;
        visible: Smallint;
        newPageBefore: Smallint;
        newPageAfter: Smallint;
        keepTogether: Smallint;
        suppressBlankSection: Smallint;
        resetPageNAfter: Smallint;
        printAtBottomOfPage: Smallint;
        backgroundColor: COLORREF;
        underlaySection: Smallint;
        showArea: Smallint;
        freeFormPlacement: Smallint;
    end;
```

PESessionInfo

PESessionInfo contains information about the current Microsoft Access session. Many Microsoft Access database tables require that a session be opened before the table can be used. Use this structure with “PEGetNthTableSessionInfo” on page 349 to retrieve current information and “PESetNthTableSessionInfo” on page 429 to pass new information.

C Syntax

```
typedef struct PESessionInfo {
    WORD StructSize;
    char UserID [PE_SESS_USERID_LEN];
    char Password [PE_SESS_PASSWORD_LEN];
    DWORD SessionHandle;
};
```

Members

StructSize	Specifies the size of the PESessionInfo structure. Initialize this member to PE_SIZEOF_SESSION_INFO.
UserID	Specifies the user ID value (of length PE_SESS_USERID_LEN = 128) needed for logging on to the system.
Password	Specifies the password (of length PE_SESS_PASSWORD_LEN = 128) needed for logging on to the system. Password is undefined when getting information from a report.
SessionHandle	The handle to the current MS Access session. When setting information, if SessionHandle = 0, the UserID and Password settings are used; otherwise the SessionHandle is used. SessionHandle is undefined when getting information from a report.

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 Password member before calling “PESetNthTableSessionInfo” on page 429.
- If the Access database contains database-level security, use a newline character, ‘\n’ (ASCII character 10) followed by the database-level password (for example, “\ndbpassword”).
- If the Access database contains both session security and database-level security, use the session password followed by the newline character and the database password (for example, “sesspswd\ndbpassword”).
- Alternately, database-level security can also be handled by assigning the database-level password to the Password member of “PELogOnInfo” on page 479 and calling “PESetNthTableLogOnInfo” on page 427.

VB Type Listing

```
Type PESessionInfo
    StructSize As Integer
    UserID As String * PE_SESS_USERID_LEN
    Password As String * PE_SESS_PASSWORD_LEN
    SessionHandle As Long
End Type
```

Delphi Record Listing

```
type
    PESesPassType = array[1..PE_SESS_PASSWORD_LEN] of char;
    PESessionInfo = record
        StructSize: Word;
        UserID: PESesPassType;
        Password: PESesPassType;
        SessionHandle: DWord;
    end;
```

PEShowGroupEventInfo

PEShowGroupEventInfo contains information when an event callback is called with event ID equal to PE_SHOW_GROUP_EVENT.

C Syntax

```
typedef struct PEShowGroupEventInfo {
    WORD StructSize;
    WORD groupLevel;
    long windowHandle;
    char **groupList;
}PEShowGroupEventInfo;
```

Members

StructSize	Specifies the size of the PEShowGroupEventInfo structure. Initialize this member to PE_SIZEOF_SHOW_GROUP_EVENT_INFO.
groupLevel	The number of the group name in the group list.
windowHandle	Frame window handle where the event happens.
groupList	Specifies a pointer to an array of group names describing the group path to which the group area is going to navigate. This pointer is freed after the callback function.

Remarks

Make a copy of the groupList if you want to keep group path information. CRPE frees the groupList after the callback function.

Delphi Record Listing

```
type
    PEPCharPointer = ^PChar;
    PEShowGroupEventInfo = record
        StructSize: Word;
        groupLevel: Word;
        windowHandle: Longint;
        groupList: PEPCharPointer;
    end;
```

PEStartEventInfo

PEStartEventInfo contains start event information when the callback function is called with event ID equal to PE_START_EVENT.

C Syntax

```
typedef struct PEStartEventInfo {
    WORD StructSize;
    WORD destination;
}PEStartEventInfo;
```

Members

StructSize	Specifies the size of the PEStartEventInfo structure. Initialize this member to PE_SIZEOF_START_EVENT_INFO.
destination	Specifies the process destination. Uses one of the PE_TO_XXX "Job Destination Constants" on page 557.

VB Type Listing

```
Type PEStartEventInfo
    StructSize As Integer
    destination As Integer
End Type
```

Delphi Record Listing

```
type
    PEStartEventInfo = record
        StructSize: Word;
        destination: Word;
    end;
```

PEStopEventInfo

PEStopEventInfo contains stop event information when a callback function is called with an event ID equal to PE_STOP_EVENT.

C Syntax

```
typedef struct PEStopEventInfo {
    WORD StructSize;
    WORD destination;
    WORD jobStatus;
} PEStopEventInfo;
```

Members

StructSize	Specifies the size of the PEStopEventInfo structure. Initialize this member to PE_SIZEOF_STOP_EVENT_INFO.
destination	Specifies the process destination. Uses one of the PE_TO_XXX "Job Destination Constants" on page 557.
jobStatus	Indicates the current status of the job. Uses one of the PE_JOBXXX "Job Status Constants" on page 558.

VB Type Listing

```
Type PEStopEventInfo
    StructSize As Integer
    destination As Integer
    jobStatus As Integer
End Type
```

Delphi Record Listing

```
type
    PEStopEventInfo  = record
        StructSize: Word;
        destination: Word;
        jobStatus: Word;
    end;
```

PESubreportInfo

PESubreportInfo contains information about subreports in a report. This structure is used by "PEGetSubreportInfo" on page 367, to gather information about a specified subreport.

C Syntax

```
typedef struct PESubreportInfo {
    WORD StructSize;
    char name [PE_SUBREPORT_NAME_LEN];
    short NLinks;
    short isOnDemand;
    short external;
    short reimportOption;
} PESubreportInfo;
```

Members

StructSize	Specifies the size of the PESubreportInfo structure. Initialize this member to PE_SIZEOF_SUBREPORT_INFO.
name	Specifies the string (of length PE_SUBREPORT_NAME_LEN = 128, NULL-terminated) containing the name of the subreport. This is the name assigned to the subreport when it was first created.
NLinks	Specifies the number of links to primary report data.
isOnDemand	TRUE if the subreport is a real-time subreport. FALSE otherwise.
external	TRUE if the subreport is imported. FALSE otherwise.
reimportOption	Specifies the update option for the subreport. Use PE_SRI_ONOPENJOB (reimport the subreport when the main report is opened) or PE_SRI_ONFUNCTIONCALL (reimport the subreport when the API is called).

VB Type Listing

```
Type PESubreportInfo
    StructSize As Integer
    Name As String * PE_SUBREPORT_NAME_LEN
    NLinks As Integer
    IsOnDemand As Integer
    external As Integer
    reimportOption As Integer
End Type
```

Delphi Record Listing

```
type
    PESubreportNameType
    = array[0..PE_SUBREPORT_NAME_LEN-1] of char;
    PESubreportInfo = record
        structSize: Word;
        name: PESubreportNameType;
        NLinks: smallint
        IsOnDemand: smallint
    end;
```

PETableDifferenceInfo

Read-Only Struct PETableDifferenceInfo contains database table information that is used by “PECheckNthTableDifferences” on page 284.

C Syntax

```
typedef struct PETableDifferenceInfo {
    WORD StructSize;
    DWORD tableDifferences;
    DWORD reserved1;
    DWORD reserved2;
} PETableDifferenceInfo;
```

Members

StructSize	Specifies the size of the PETableLocation structure. Initialize this member to PE_SIZEOF_TABLE_DIFFERENCE_INFO.	
tableDifferences	Read-Only. Returns any combination of the following PE_TCD_XXX TableDifference Constants.	
	Constant	**Value**
	PE_TCD_OKAY	0x00000000
	PE_TCD_DATABASENOTFOUND	0x00000001
	PE_TCD_SERVERNOTFOUND	0x00000002
	PE_TCD_SERVERNOTOPENED	0x00000004
	PE_TCD_ALIASCHANGED	0x00000008
	PE_TCD_INDEXESCHANGED	0x00000010
	PE_TCD_DRIVERCHANGED	0x00000020
	PE_TCD_DICTIONARYCHANGED	0x00000040
	PE_TCD_FILETYPECHANGED	0x00000080
	PE_TCD_RECORDSIZECHANGED	0x00000100
	PE_TCD_ACCESSCHANGED	0x00000200
	PE_TCD_PARAMETERSCHANGED	0x00000400
	PE_TCD_LOCATIONCHANGED	0x00000800
	PE_TCD_DATABASEOTHER	0x00001000
	PE_TCD_NUMFIELDSCHANGED	0x00010000
	PE_TCD_FIELDOTHER	0x00020000
	PE_TCD_FIELDNAMECHANGED	0x00040000
	PE_TCD_FIELDDESCCHANGED	0x00080000
	PE_TCD_FIELDTYPECHANGED	0x00100000
	PE_TCD_FIELDSIZECHANGED	0x00200000
	PE_TCD_NATIVEFIELDTYPECHANGED	0x00400000
	PE_TCD_NATIVEFIELDOFFSETCHANGED	0x00800000
	PE_TCD_NATIVEFIELDSIZECHANGED	0x01000000
	PE_TCD_FIELDDECPLACESCHANGED	0x02000000
reserved1	Reserved. Do not use.	
reserved2	Reserved. Do not use.	

VB Type Listing

```
Type PETableDifferenceInfo
    StructSize As Integer
    tableDifferences As Long
    reserved1 As Long
    reserved2 As Long
End Type
```

Delphi Record Listing

```
type
  PETableDifferenceInfo = record
    StructSize       : Word;
    tableDifferences : DWord;
    reserved1        : DWord;
    reserved2        : DWord;
  end;
```

PETableLocation

PETableLocation contains database location information that is used with the "PEGetNthTableLocation" on page 347 to gather current location information and "PESetNthTableLocation" on page 426 to pass new location information.

C Syntax

```
typedef struct PETableLocation {
    WORD StructSize;
    char Location [PE_TABLE_LOCATION_LEN];
    char SubLocation[PE_TABLE_LOCATION_LEN];
    char ConnectBuffer[PE_CONNECTION_BUFFER_LEN];
} PETableLocation;
```

Members

StructSize	Specifies the size of the PETableLocation structure. Initialize this member to PE_SIZEOF_TABLE_LOCATION.
Location	Specifies the database location (of length PE_TABLE_LOCATION_LEN = 256, NULL-terminated). This member is database dependent and must be formatted correctly for the expected database. The following table lists some examples.
	Database Location Examples
	xBASE (Natively): <drive>:\<path>\<file>
	xBASE (ODBC): <datasource name>
	Paradox (Natively): <drive>:\<path>\<file>
	Paradox (ODBC): <datasource name>
	Btrieve (Natively): <drive>:\<path>\<file>
	Btrieve (ODBC): <datasource name>

	Oracle (Natively): <database>.<table>
	Oracle (ODBC): <database>.<table>
	SQL Server (Natively): <database>.<owner>.<table>
	SQL Server (ODBC): <database>.<owner>.<table>
SubLocation	Specifies the database sublocation (of length PE_TABLE_LOCATION_LEN = 256, NULL-terminated).
ConnectBuffer	Specifies the connection buffer (of length PE_CONNECTION_BUFFER_LEN = 512, NULL-terminated) containing connection information for attached tables.

VB Type Listing

```
Type PETableLocation
    StructSize As Integer
    Location  As String * PE_TABLE_LOCATION_LEN
    SubLocation As String * PE_TABLE_LOCATION_LEN
    ConnectBuffer As String * PE_CONNECTION_BUFFER_LEN
End Type
```

Delphi Record Listing

```
type
  PETableLocType = array [0..PE_TABLE_LOCATION_LEN - 1] of Char;
  PEConnectBufferType = array [0..PE_CONNECTION_BUFFER_LEN - 1] of Char;
  PETableLocation = record
    StructSize    : Word;
    Location      : PETableLocType;
    SubLocation   : PETableLocType; { For MS Access Table Names }
    ConnectBuffer : PEConnectBufferType;
  end;
```

PETablePrivateInfo

PETablePrivateInfo contains information for using data objects such as ADO, RDO, or CDO with the Active Data Driver(PS2MON.DLL).

C Syntax

```
typedef struct PETablePrivateInfo {
    WORD StructSize;
    WORD nBytes;
    DWORD tag;
    BYTE FAR *dataPtr;
} PETablePrivateInfo;
```

Members

StructSize	Specifies the size of the PETablePrivateInfo structure. Initialize this member to PE_SIZEOF_TABLE_PRIVATE_INFO.
nBytes	Specifies the length of the data starting at the dataPtr.
tag	Specifies a value indicating the type of data being passed to theDatabaseTable 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.
dataPtr	Specifies a pointer to variant data passed to the database driver. With Active data, this must be a Recordset object if DAO, ADO, or the Visual Basic data control is used. If CDO is used, this must be a Rowset object.

Delphi Record Listing

```
type
    crBytePointer = ^Byte;
    PETablePrivateInfo = record
                StructSize: Word;  {initialize to
                            PE_SIZEOF_TABLE_PRIVATE_INFO}
                    nBytes: Smallint;
                       tag: DWORD;
                   dataPtr: crBytePointer;
    end;
```

PETableType

PETableType contains information for identifying the type of a specified table. This information is gathered using “PEGetNthTableType” on page 350.

C Syntax

```
typedef struct PETableType {
    WORD StructSize;
    char DLLName [PE_DLL_NAME_LEN];
    char DescriptiveName [PE_FULL_NAME_LEN];
    WORD DBType;
} PETableType;
```

Members

<table>
<tr><td>StructSize</td><td colspan="2">Specifies the size of the PETableType structure. Initialize this member to PE_SIZEOF_TABLE_TYPE.</td></tr>
<tr><td>DLLName</td><td colspan="2">Specifies the name of the appropriate database DLL (of length PE_DLL_NAME_LEN = 64, NULL-terminated) for the table of interest. Select the DLL you want to use from the table below:</td></tr>
<tr><td></td><td>Use this DLL</td><td>For this standard non-SQL database</td></tr>
<tr><td></td><td>PDBBDE.DLL</td><td>Borland Database Engine</td></tr>
<tr><td></td><td>PDBBND.DLL</td><td>Bound reports</td></tr>
</table>

	PDBDAO.DLL	DAO data sources (Access)
	PDBJET.DLL	Access
	PDBPDX.DLL	Paradox
	PDBXBSE.DLL	dBASE, FoxPro, Clipper
	PDCTBTRV.DLL	Btrieve
	PDSDB22.DLL	DB2/2
	PDSGUPTA.DLL	Gupta
	PDSNETW.DLL	Netware
	PDSODBC.DLL	ODBC. See Remarks below.
	PDSORACL.DLL	Oracle
	PDSSYB10.DLL	Sybase 10/11
	PDSSYBAS.DLL	Sybase
DescriptiveName	Specifies the full description of the table of interest (of length PE_FULL_NAME_LEN = 256, NULL-terminated).	
DBType	Specifies the type of database that contains the table of interest. Use one of the PE_DT_XXX "Database Type Constants" on page 545.	

Remarks

For PDSODBC.DLL, the DescriptiveName includes the ODBC data source name.

VB Type Listing

```
Type PETableType
    StructSize As Integer
    DLLName As String * PE_DLL_NAME_LEN
    DescriptiveName As String * PE_FULL_NAME_LEN
    DBType As Integer
End Type
```

Delphi Record Listing

```
type
    PEDllNameType = array[0..PE_DLL_NAME_LEN-1] of char;
    PEFullNameType = array[0..PE_FULL_NAME_LEN-1] of char;
    PETableType = record
        StructSize: Word;
        DLLName: PEDllNameType;
        DescriptiveName: PEFullNameType;
        DBType: Word;
    end;
```

PETrackCursorInfo

PETrackCursorInfo is used to retrieve or set the track cursor information for the preview window by using "PEGetTrackCursorInfo" on page 368, and "PESetTrackCursorInfo" on page 444.

C Syntax

```
typedef struct PETrackCursorInfo {
    WORD StructSize
    short groupAreaCursor;
    short groupAreaFieldCursor;
    short detailAreaCursor;
    short detailAreaFieldCursor;
    short graphCursor;
    long groupAreaCursorHandle;
    long groupAreaFieldCursorHandle;
    long detailAreaCursorHandle;
    long detailAreaFieldCursorHandle;
    long graphCursorHandle;
    short ondemandSubreportCursor;
    short hyperlinkCursor;
} PETrackCursorInfo;
```

Members

- Each member of type short can be set to one of the PE_TCD_XXX "Track Cursor Constants" on page 560, or PE_UNCHANGED for no change.
- Each member of type long is currently reserved and should not be used.

StructSize	Specifies the size of the PETrackCursorInfo structure. Initialize this member to PE_SIZEOF_TRACK_CURSOR_INFO.
groupAreaCursor	Specifies the group area cursor. Use one of the PE_TC_XXX "Track Cursor Constants" on page 560 or PE_UNCHANGED for no change.
groupAreaFieldCursor	Specifies the cursor for a memo, blob, database, summary, or formula field in the group area. Use one of the PE_TC_XXX "Track Cursor Constants" on page 560 or PE_UNCHANGED for no change.
detailAreaCursor	Specifies the Details area cursor. Use one of the PE_TC_XXX "Track Cursor Constants" on page 560 or PE_UNCHANGED for no change.
detailAreaFieldCursor	Specifies the cursor for a memo, blob, database, summary, or formula field in the Details area. Use one of the PE_TC_XXX "Track Cursor Constants" on page 560 or PE_UNCHANGED for no change.
graphCursor	Specifies the cursor for the group chart in the Report Header or Report Footer area. Use one of the PE_TC_XXX "Track Cursor Constants" on page 560 or PE_UNCHANGED for no change.
groupAreaCursor Handle	Reserved, do not use.
groupAreaFiedlCursor Handle	Reserved, do not use.

detailAreaCursor Handle	Reserved, do not use.
detailAreaFieldCursor Handle	Reserved, do not use.
graphCursorHandle	Reserved, do not use.
ondemandSubreport Cursor	Specifies the cursor for on-demand subreports when drilldown for the window is enabled. Default is PE_TC_MAGNIFY_CURSOR
hyperlinkCursor	Specifies the cursor for hyperlink text in report objects. Default is PE_TC_HAND_CURSOR.

Remarks

- By default, all the cursors are PE_TC_ARROW_CURSOR. If the canDrillDown option in "PEWindowOptions" on page 519, is True, then groupAreaCursor, groupAreaFieldCursor, and graphCursor will be PE_TC_MAGNIFY_CURSOR.

VB Type Listing

```
Type PETrackCursorInfo
    groupAreaCursor As Integer
    groupAreaFieldCursor As Integer
    detailAreaCursor As Integer
    detailAreaFieldCursor As Integer
    graphCursor As Integer
    groupAreaCursorHandle As Long
    groupAreaFieldCursorHandle As Long
    detailAreaCursorHandle As Long
    detailAreaFieldCursorHandle As Long
    graphCursorHandle As Long
End Type
```

Delphi Record Listing

```
type
    PETrackCursorInfo = record
        StructSize: Word;
        groupAreaCursor: smallint;
        groupAreaFieldCursor: smallint;
        detailAreaCursor: smallint;
        detailAreaFieldCursor: smallint;
        graphCursor: smallint;
        groupAreaCursorHandle:longint;
        groupAreaFieldCursorHandle: longint;
        detailAreaCursorHandle: longint;
        detailAreaFieldCursorHandle: longint;
        graphCursorHandle: longint; {
    end;
```

PEValueInfo

PEValueInfo contains information that is used by “PEConvertPFInfotoVInfo” on page 290 to return converted parameter values in simple types and by “PEConvertVInfotoPFInfo” on page 291 to accept values for conversion to the binary format required by “PESetNthParameterField” on page 423.

C Syntax

```
typedef struct PEValueInfo {
    WORD StructSize;
    WORD valueType;
    double viNumber;
    double viCurrency;
    BOOL viBoolean;
    char viString[PE_VI_STRING_LEN];
    short viDate[3];
    short viDateTime[6];
    short viTime[3];
    COLORREF viColor;
    short viInteger;
    char viC;
    char ignored;
    long viLong
} PEValueInfo;
```

Members

StructSize	Specifies the size of the PEValueInfo structure. Initialize this member to PE_SIZEOF_VALUE_INFO.	
valueType	Specifies the data type of the parameter field. The Crystal Report Engine supports the following data types and associated PE_VI_XXX Value Type constants.	
	Data Type	**Constant**
	Number	PE_VI_NUMBER
	Currency	PE_VI_CURRENCY
	Boolean	PE_VI_BOOLEAN
	Date	PE_VI_DATE
	String	PE_VI_STRING
	DateTime	PE_VI_DATETIME
	Time	PE_VI_TIME

	Integer	PE_VI_INTEGER
	COLOREF	PE_VI_COLOR
	Char	PE_VI_CHAR
	Long	PE_VI_LONG
	No Value	PE_VI_NOVALUE
viNumber	Specifies the value if the parameter is a numeric value.	
viCurrency	Specifies the value if the parameter is a currency value.	
viBoolean	Specifies the value if the parameter is a Boolean value.	
viString	Specifies the value if the parameter is a string value (of length PE_VI_STRING_LEN = 256).	
viDate	Specifies the value if the parameter is a Date value (year, month, day).	
viDateTime	Specifies the value if the parameter is a Date/Time value (year, month, day, hour, minute, second).	
viTime	Specifies the value if the parameter is a Time value (hour, minute, second).	
viColor	Specifies the value if the parameter is a color value.	
viInteger	Specifies the value if the parameter is a integer value.	
viC	Specifies the value if the parameter is a char value.	
ignored	Internal use only for 4 byte alignment. Do not use.	
viLong	Specifies the value if the parameter is a long value.	

VB Type Listing

```
Type PEValueInfo
    StructSize As Integer
    valueType As Integer
    viNumber As Double
    viCurrency As Double
    viBoolean As Long
    viString As String * PE_VI_STRING_LEN
    viDate(0 To 2) As Integer
    viDateTime(0 To 5) As Integer
    viTime(0 To 2) As Integer
    viColor As Long
    viInteger As Integer
    viC As Byte
    ignored As Byte
    viLong As Long
End Type
```

Delphi Record Listing

```
type
    PEVALUEINFOSTRINGTYPE
    = array[0..PE_VI_STRING_LEN-1] of smallint;
    PEVALUEINFODATEORTIMETYPE = array[0..5] of smallint;
    PEVALUEINFODATETIMETYPE = array[0..2] of smallint;
    PEValueInfo = record
        StructSize:Word;
        valueType: Word; {a PE_VI_constant
        viNumber: Double;
        viCurrency: Double;
        viBoolean: BOOL;
        viString: PEVALUEINFOSTRINGTYPE;
        viDate: PEVALUEINFODATEORTIMETYPE;
        viDateTime: PEVALUEINFODATETIMETYPE;
        viTime: PEVALUEINFODATEORTIMETYPE;
        viColor: COLORREF;
        viInteger: Smallint;
        viC: Char; Char;{BYTE}
        ignored: Char;
        viLong: Longint;
    end;
```

PEVersionInfo

PEVersionInfo contains information on the version number of the report associated with the job handle. This information is used by "PEGetReportVersion" on page 359, to retrieve major and minor version information on the report.

C Syntax

```
typedef struct PEVersionInfo
{
    WORD StructSize;
    WORD major;
    WORD minor;
    char letter;
} PEVersionInfo;
```

Members

StructSize	Specifies the size of the PEVersionInfo structure. Initialize this member to PE_SIZEOF_VERSION_INFO.
major	Specifies the major release number of the report.
minor	Specifies the minor release number of the.report.
letter	Specifies a character value associated with the minor version number of the report. Most reports do not have a character value, so a NULL value will be returned.

VB Type Listing

```
Type PEVersionInfo
    StructSize As Integer
    major As Integer
    minor As Integer
    letter As String
End Type
```

Delphi Record Listing

```
type PEVersionInfo = record
    StructSize : Word;
    major : Word;
    minor : Word;
    character : PChar;
end;
```

PEWindowOptions

PEWindowOptions contains information related to preview window options. This structure is used by “PEGetWindowOptions” on page 371, and “PESetWindowOptions” on page 445, to get and set preview window contents.

C Syntax

```
typedef struct PEWindowOptions {
    WORD StructSize;
    short hasGroupTree;
    short canDrillDown;
    short hasNavigationControls;
    short hasCancelButton;
    short hasPrintButton;
    short hasExportButton;
    short hasZoomControl;
    short hasCloseButton;
    short hasProgressControls;
    short hasSearchButton;
    short hasPrintSetupButton;
    short hasRefreshButton;
    short showToolbarTips;
    short showDocumentTips;
short hasLaunchButton;
} PEWindowOptions;
```

Members

Each member of type short can be set to TRUE, FALSE, or PE_UNCHANGED for no change.

StructSize	Specifies the size of the PEWindowOptions structure. Initialize this member to PE_SIZEOF_WINDOW_OPTIONS.
hasGroupTree	Specifies whether or not the Group Tree will appear in the window. Whether or not the preview window has a group tree is determined by two flags. One is this option (hasGroupTree) and the other is the hasGroupTreeOption in Crystal Reports (File | Report Options dialog box). By default, this value is set to False.
canDrillDown	Specifies whether or not drill-down capabilities will be active in the window. By default, this value is set to False.
hasNavigation Controls	Specifies whether or not navigation controls will appears in the window. By default, this value is set to True.
hasCancelButton	Specifies whether or not a Cancel button will appear in the window. By default, this value is set to True.
hasPrintButton	Specifies whether or not a Print button will appear in the window. By default, this value is set to True.
hasExportButton	Specifies whether or not an Export button will appear in the window By default, this value is set to True.
hasZoomControl	Specifies whether or not zoom controls will appear in the window. By default, this value is set to True.
hasCloseButton	Specifies whether or not a Close button will appear in the window. By default, this value is set to False. If canDrillDown is set to True, CRPE will turnhasCloseButton on unless you set it to False.
hasProgressControls	Specifies whether or not progress controls will appear in the window. By default, this value is set to True.
hasSearchButton	Specifies whether or not a Search button will appear in the window. By default, this value is set to False.
hasPrintSetupButton	Specifies whether or not a Print Setup button will appear in the window.
hasRefreshButton	Specifies whether or not a Refresh button will appear in the window.
showToolbarTips	Specifies whether or not Tooltips are shown on the Toolbar. By default, this value is set to True (default is visible tooltips on toolbar).
showDocumentTips	Specifies whether or not Tooltips are shown on the Document. By default, this value is set to False (default is hidden tooltips on document).
hasLaunchButton	Specifies whether or not a button to launch Seagate Analysis is placed on the toolbar. By default, this value is set to False. **Note:** Seagate Analysis is now known as Crystal Analysis.

VB Type Listing

```
Type PEWindowOptions
    StructSize As Integer
    hasGroupTree As Integer
    canDrillDown As Integer
    hasNavigationControls As Integer
    hasCancelButton As Integer
    hasPrintButton As Integer
    hasExportButton As Integer
    hasZoomControl As Integer
    hasCloseButton As Integer
    hasProgressControls As Integer
    hasSearchButton As Integer
    hasPrintSetupButton As Integer
    hasRefreshButton As Integer
    showToolbarTips As Integer
    showDocumentTips As Integer
End Type
```

Delphi Recor.d Listing

```
type
    PEWindowOptions = record
        StructSize: Word;
        hasGroupTree: Smallint;
        canDrillDown: Smallint;
        hasNavigationControls: Smallint;
        hasCancelButton: Smallint;
        hasPrintButton: Smallint;
        hasExportButton: Smallint;
        hasZoomControl: Smallint;
        hasCloseButton: Smallint;
        hasProgressControls: Smallint;
        hasSearchButton: Smallint;
        hasPrintSetupButton: Smallint;
        hasRefreshButton: Smallint;
        showToolbarTips: Smallint;
        showDocumentTips:  Smallint;
    end;
```

PEZoomLevelChangingEventInfo

PEZoomLevelChangingEventInfo contains information about records read when a callback function is called with event ID equal to PE_ZOOM_LEVEL_CHANGING_EVENT.

C Syntax

```
typedef struct PEZoomLevelChangingEventInfo {
    WORD StructSize;
    WORD zoomLevel;
    long windowHandle;
} PEZoomLevelChangingEventInfo;
```

Members

StructSize	Specifies the size of the PEZoomLevelChangingEventInfo structure. Initialize this member to PE_SIZEOF_ZOOM_LEVEL_CHANGNG_EVENT_INFO.
zoomLevel	The zoom level set in the preview window. This can be a value from 25 to 400, indicating a magnification percentage, or it can be one of the "Zoom Level Constants" on page 561.
windowHandle	Specifies the frame window handle where the event happens.

VB Type Listing

```
Type PEZoomLevelChangingEventInfo
    StructSize As Integer
    zoomLevel As Integer
    windowHandle As Long
End Type
```

Delphi Record Listing

```
type
    PEZoomLevelChangingEventInfo = record
        StructSize: Word;
        zoomLevel: Word;
        windowHandle: HWnd;
    end;
```

UXDDiskOptions

UXDDiskOptions contains file name information that is used by "PEEnableEventInfo" on page 457, when you want to export to a disk file.

C Syntax

```
struct UXDDiskOptions {
    WORD StructSize;
    char FAR *fileName;
};
```

Members

structSize	Specifies the size of the UXDDiskOptions structure. You must initialize this member to be the size of whatever it is, for example, options.structSize = UXDDiskOptionsSize.
fileName	Specifies the pointer to the null-terminated string that contains the file name under which you want your disk file saved.

Delphi Record Listing

```
type
    UXDDiskOptions = record
        structSize: Word;
        fileName: PChar;
    end;
```

UXDMAPIOptions

UXDMAPIOptions contains e-mail information that is used by “PEEnableEventInfo” on page 457, when you want to export to a MAPI e-mail destination.

C Syntax

```
struct UXDMAPIOptions {
    WORD StructSize;
    char FAR *toList;
    char FAR *ccList;
    char FAR *subject;
    char FAR *message;
    WORD nRecipients;
    lpMapiRecipDesc recipients;
}
```

Members

structSize	Specifies the size of the UXDMAPIOptions structure. You must initialize this member to be the size of whatever it is, for example, options.structSize = UXDMAPIOptionsSize.
toList	Specifies the pointer to the null-terminated string that contains the “To” list to which you want your e-mail message directed. If you specify “recipients” in this structure, “toList” is ignored.
ccList	Specifies the pointer to the null-terminated string that contains the “CC” list to which you want your e-mail message copied. This string will appear on the message. If you specify “recipients” in this structure, “ccList” is ignored.
subject	Specifies the pointer to the null-terminated string you want to appear as the subject line in the e-mail message.
message	Specifies the pointer to the null-terminated string you want to appear as the body of your e-mail message.
nRecipients	Specifies the number of recipients that are to receive the e-mail message. You must pass the value “0” if you specify “To” and “CC” lists in this structure.
recipients	Specifies the pointer to an array of structures, each of which describes someone to whom the message is being sent or CC'd. This member is included for those applications that are already using Microsoft's MAPI.DLL. If you are not using MAPI.DLL in your application, there is no advantage to using the recipients array over the toList and ccList. You must pass the value “0” if you specify “To” and “CC” lists in this structure. For detailed information about this member, please refer to Microsoft's MAPI documentation.

Delphi Record Listing

```
type
    UXDMAPIOptions   = record
        structSize: Word;
        toList: PChar;
        ccList: PChar;
        subject: PChar;
        message: PChar;
    end;
```

UXDSMIOptions

UXDSMIOptions contains e-mail information that is used by the "PEEnableEventInfo" on page 457, when you want to export to a MAPI e-mail destination.

C Syntax

```
struct UXDSMIOptions {
    WORD StructSize;
    char FAR *toList;
    char FAR *ccList;
    char FAR *subject;
    char FAR *message;
}
```

Members

structSize	Specifies the size of the UXDMAPIOptions structure. You must initialize this member to be the size of whatever it is, for example, options.structSize = UXDMAPIOptionsSize.
toList	Specifies the pointer to the null-terminated string that contains the "To" list to which you want your e-mail message directed. If you specify "recipients" in this structure, "toList" is ignored.
ccList	Specifies the pointer to the null-terminated string that contains the "CC" list to which you want your e-mail message copied. This string will appear on the message. If you specify "recipients" in this structure, "ccList" is ignored.
subject	Specifies the pointer to the null-terminated string you want to appear as the subject line in the e-mail message.
message	Specifies the pointer to the null-terminated string you want to appear as the body of your e-mail message.

Delphi Record Listing

```
type
    UXDSMIOptions   = record
        structSize: Word;
        toList: PChar;
        ccList: PChar;
        subject: PChar;
        message: PChar;
    end;
```

UXDPostFolderOptions

UXDPostFolderOptions contains information that is used by "PEEnableEventInfo" on page 457, when you want to export to Microsoft Exchange.

C Syntax

```
struct UXDPostFolderOptions {
    WORD StructSize;
    LPSTR pszProfile;
    LPSTR pszPassword;
    WORD wDestType;
    LPSTR pszFolderPath;
};
```

Members

<table>
<tr><td>structSize</td><td colspan="3">Specifies the size of the UXDPostFolderOptions structure. You must initialize this member to be the size of whatever it is, for example, options.structSize = UXDPostFolderOptionsSize.</td></tr>
<tr><td>pszProfile</td><td colspan="3">Specifies the Exchange profile.</td></tr>
<tr><td>pszPassword</td><td colspan="3">Specifies the Exchange password.</td></tr>
<tr><td>wDestType</td><td colspan="3">Specifies the type of report to export to. Use one of the following:</td></tr>
<tr><td></td><td>Constant</td><td>Value</td><td>Description</td></tr>
<tr><td></td><td>UXDExchFolderType</td><td>0</td><td>wDestType for Microsoft Exchange folder.</td></tr>
<tr><td></td><td>UXDPostDocMessage</td><td>1009</td><td>wDestType for folder messages.</td></tr>
<tr><td></td><td>UXDPostPersonalReport</td><td>1010</td><td>wDestType for personal report.</td></tr>
<tr><td></td><td>UXDPostFolderReport</td><td>1011</td><td>wDestType for folder report.</td></tr>
<tr><td>pszFolderPath</td><td colspan="3">Specifies the Exchange path where you want the program to place the exported file.</td></tr>
</table>

Delphi Record Listing

```
type
    UXDPostFolderOptions = record
        structSize:Word;
        pszProfile:PChar;
        pszPassword:PChar;
        wDestType: Word;
        pszFolderPath:PChar;
    end;
```

(*pszFolderPath has to be in the following format: <Message Store Name>@<Folder Name>@<Folder Name>*)

UXDVIMOptions

UXDVIMOptions contains e-mail message information that is used by "PEEnableEventInfo" on page 457, when you want to export a VIM e-mail destination.

C Syntax

```
struct UXDVIMOptions {
    WORD StructSize;
    char FAR *toList;
    char FAR *ccList;
    char FAR *bccList
    char FAR *subject;
    char FAR *message;
};
```

Note: VIM is not supported in the 32-bit version of Crystal Reports.

Members

structSize	Specifies the size of the UXDVIMOptions structure. You must initialize this member to be the size of whatever it is, for example, options.structSize = UXDVIMOptionsSize.
toList	Specifies the pointer to the null-terminated string that contains the "To" list to which you want your e-mail message directed.
ccList	Specifies the pointer to the null-terminated string that contains the "CC" list to which you want your e-mail message copied. This string will appear on the message.
bccList	Specifies the pointer to the null-terminated string that contains the "Blind CC" list to which you want your e-mail message copied. This string will not appear on the message.
subject	Specifies the pointer to the null-terminated string you want to appear as the subject line in the e-mail message.
message	Specifies the pointer to the null-terminated string you want to appear as the body of your e-mail message.

UXFCharSeparatedOptions

UXFCharSeparatedOptions contains number and date information used by the "PEEnableEventInfo" on page 457, structure when you want to export in a Character Separated format and hard code number and / or date options.

C Syntax

```
struct UXFCharSeparatedOptions {
    WORD StructSize;
    BOOL useReportNumberFormat;
    BOOL useReportDateFormat;
    char stringDelimiter;
    char FAR *fieldDelimiter;
};
```

Members

structSize	Specifies the size of the UXFCharSeparatedOptions structure. You must initialize this member to be the size of whatever it is, for example, options.structSize = UXFCharSeparatedOptionsSize.
useReport NumberFormat	Indicates whether or not the program should save numbers in the same format (decimal places, negatives, etc.) that you have used in the report. Pass TRUE if you want the program to use the same format used in the report, FALSE if you want the number saved in a format that has been optimized for the file format you have selected.
useReportDate Format	Indicates whether or not the program should save dates in the same format (MDY, DMY, etc.) that you used in the report. Pass TRUE if you want the program to use the same format as used in the report, FALSE if you want dates saved in a format that has been optimized for the file format you have selected.
stringDelimiter	Specifies the character you want to use to enclose alphanumeric field data in the character separated values format. You can use whatever character you wish, and it must be enclosed in quotes.
fieldDelimiter	Specifies a pointer to the string you want to use to separate the fields in the character separated values format. Your string may be up to 16 characters long and must be enclosed in quotes.

Delphi Record Listing

```
type
    UXFCharSeparatedOptions = record
        structSize: Word;
        useReportNumberFormat: Bool;
        useReportDateFormat: Bool;
        stringDelimiter: Char;
        fieldDelimiter: PChar;
    end;
```

UXFCommaTabSeparatedOptions

UXFCommaTabSeparatedOptions contains number and date information used by "PEEnableEventInfo" on page 457, when you want to export in a Comma-separated or Tab-separated format and hard code number and/or date options.

C Syntax

```
struct UXFCommaTabSeparatedOptions {
    WORD StructSize;
    BOOL useReportNumberFormat;
    BOOL useReportDateFormat;
};
```

Members

structSize	Specifies the size of the UXFCommaTabSeparatedOptions structure. You must initialize this member to be the size of whatever it is, for example, options.structSize = UXFCommaTabSeparatedOptionsSize.
useReport NumberFormat	Indicates whether or not the program should save numbers in the same format (decimal places, negatives, etc.) that you have used in the report. Pass TRUE if you want the program to use the same format used in the report, FALSE if you want the number saved in a format that has been optimized for the file format you have selected.
useReportDate Format	Indicates whether or not the program should save dates in the same format (MDY, DMY, etc.) that you used in the report. Pass TRUE if you want the program to use the same format as used in the report, FALSE if you want dates saved in a format that has been optimized for the file format you have selected.

Delphi Record Listing

```
type
    UXFCommaTabSeparatedOptions = record
        structSize: Word;
        useReportNumberFormat: Bool;
        useReportDateFormat: Bool;
    end;
```

UXFDIFOptions

UXFDIFOptions contains number and date information used by "PEEnableEventInfo" on page 457, when you want to export in a DIF format (Data Interchange Format) and hard code number and/or date options.

C Syntax

```
struct UXFDIFOptions {
    WORD StructSize;
    BOOL useReportNumberFormat;
    BOOL useReportDateFormat;
};
```

Members

structSize	Specifies the size of the UXFDIFOptions structure. You must initialize this member to be the size of whatever it is, for example, options.structSize = UXFDIFOptionsSize.
useReport NumberFormat	Indicates whether or not the program should save numbers in the same format (decimal places, negatives, etc.) that you have used in the report. Pass TRUE if you want the program to use the same format used in the report, FALSE if you want the number saved in a format that has been optimized for the file format you have selected.
useReport DateFormat	Indicates whether or not the program should save dates in the same format (MDY, DMY, etc.) that you used in the report. Pass TRUE if you want the program to use the same format as used in the report, FALSE if you want dates saved in a format that has been optimized for the file format you have selected.

Delphi Record Listing

```
type
    UXFDIFOptions = record
        structSize: Word;
        useReportNumberFormat: Bool;
        useReportDateFormat: Bool;
    end;
```

UXFHTML3Options

UXFHTML3Options contains options used by “PEEnableEventInfo” on page 457, when you are exporting to HTML format.

C Syntax

```
struct UXFHTML3Options {
    WORD StructSize;
    char FAR *fileName;
};
```

Members

structSize	Specifies the size of the UXFHTML3Options structure. You must initialize this member to be the size of whatever it is, for example, options.structSize = UXFHTML3OptionsSize.
fileName	Specifies a null terminated file name. For example, “C\pub\docs\boxoffic\default.htm”. Any exported GIF files will be located in the same directory as this file.

Delphi Record Listing

```
type
    UXFHTML3Options = record
        structSize: Word;
        fileName: PChar;
    end;
```

UXFODBCOptions

UXFODBCOptions contains information used by “PEEnableEventInfo” on page 457, whenever you export in ODBC format.

C Syntax

```
struct UXFODBCOptions {
    WORD    structSize;
    char FAR *dataSourceName;
    char FAR *dataSourceUserID;
    char FAR *dataSourcePassword;
    char FAR *exportTableName;
};
```

Members

structSize	Specifies the size of the UXFODBCOptions structure. You must initialize this member to be the size of whatever it is, for example, options.structSize = UXFODBCOptionsSize.
dataSourceName	Specifies the name of the data source that you want to export to.
dataSourceUserID	Specifies the User ID that you need to connect to the data source.
dataSourcePassword	Specifies the Password that you need to connect to the data source.
exportTableName	Specifies the name of the table you want to export to in the data source.

Delphi Record Listing

```
type
    UXFODBCOptions = record
        structSize: Word;
        dataSourceName:PChar;
        dataSourceUserID:PChar;
        dataSourcePassword:PChar;
        exportTableName:PChar;
    end;
```

UXFPaginatedTextOptions

UXFPaginatedTextOptions contains information used by “PEEnableEventInfo” on page 457, whenever you export in paginated text format.

C Syntax

```
struct UXFPaginatedTextOptions {
    WORD StructSize;
    WORD nLinesPerPage;
};
```

Members

structSize	Specifies the size of the UXFPaginatedTextOptions structure. Initialize this member to UXFPaginatedTextOptionsSize.
nLines PerPage	Indicates the number of lines to be printed before the page break. The default is 60 lines. When the paginated text format is used with "PEGetExportOptions" on page 306, the program displays the Lines Per Page dialog box to give the user the opportunity to specify a different number if he or she wishes.

Delphi Record Listing

```
type
    UXFPaginatedTextOptions = Record
        structSize:Word;
        nLinesPerPage:Word;
    end;
```

UXFRecordStyleOptions

UXFRecordStyleOptions contains number and date information used by "PEEnableEventInfo" on page 457, when you want to export in a Record style (columns of values) format and hard code number and / or date options.

C Syntax

```
struct UXFRecordStyleOptions {
    WORD StructSize;
    BOOL useReportNumberFormat;
    BOOL useReportDateFormat;
};
```

Members

structSize	Specifies the size of the UXFRecordStyleOptions structure. Initialize this member to UXFRecordStyleOptionsSize.
useReport NumberFormat	Indicates whether or not the program should save numbers in the same format (decimal places, negatives, etc.) that you have used in the report. Pass TRUE if you want the program to use the same format used in the report, FALSE if you want the number saved in a format that has been optimized for the file format you have selected.
useReport DateFormat	Indicates whether or not the program should save dates in the same format (MDY, DMY, etc.) that you used in the report. Pass TRUE if you want the program to use the same format as used in the report, FALSE if you want dates saved in a format that has been optimized for the file format you have selected.

Delphi Record Listing

```
type
    UXFRecordStyleOptions = record
        structSize: Word;
        useReportNumberFormat: Bool;
        useReportDateFormat: Bool;
    end;
```

Microsoft Windows Structures

The COLORREF and DEVMODE structures are discussed in this section.

COLORREF

Microsoft Windows COLORREF (Windef.h) is a 32-bit value used to specify an RGB color.

Syntax

When specifying an explicit RGB color, COLORREF has the following hexadecimal form:

```
0x00bbggrr
```

where the low-order byte (rr) contains a value for the relative intensity of red, the second byte (gg) contains a value for green, and the third byte (bb) contains a value for blue. The high-order byte must be zero. The maximum value for a single byte is 0xFF.

Remarks

Complete documentation for the COLORREF Microsoft Windows value can be found at
http: / / msdn.microsoft.com /

DEVMODE

The Microsoft Windows DEVMODE structures contain information about the device initialization and environment of a printer.

C Syntax

```
typedef struct _devicemode {
    BCHAR  dmDeviceName[CCHDEVICENAME];
    WORD   dmSpecVersion;
    WORD   dmDriverVersion;
    WORD   dmSize;
    WORD   dmDriverExtra;
    DWORD  dmFields;
    union {
       struct {
        short dmOrientation;
        short dmPaperSize;
        short dmPaperLength;
        short dmPaperWidth;
       };
       POINTL dmPosition;
    };
    short  dmScale;
```

```
    short  dmCopies;
    short  dmDefaultSource;
    short  dmPrintQuality;
    short  dmColor;
    short  dmDuplex;
    short  dmYResolution;
    short  dmTTOption;
    short  dmCollate;
    BCHAR  dmFormName[CCHFORMNAME];
    WORD  dmLogPixels;
    DWORD  dmBitsPerPel;
    DWORD  dmPelsWidth;
    DWORD  dmPelsHeight;
    DWORD  dmDisplayFlags;
    DWORD  dmDisplayFrequency;
  #if(WINVER >= 0x0400)
    DWORD  dmICMMethod;
    DWORD  dmICMIntent;
    DWORD  dmMediaType;
    DWORD  dmDitherType;
    DWORD  dmReserved1;
    DWORD  dmReserved2;
  #if (WINVER >= 0x0500) || (_WIN32_WINNT >= 0x0400)
    DWORD  dmPanningWidth;
    DWORD  dmPanningHeight;
  #endif /* WINVER >= 0x0500 */
  #endif /* WINVER >= 0x0400 */
} DEVMODE;
```

Members

dmDeviceName	Specifies the name of the device the driver supports (for example, PCL/HP LaserJet for PCL/HP LaserJet®). This string is unique among device drivers.
dmSpecVersion	Specifies the version number of the initialization data specification on which the structure is based.
dmDriverVersion	Specifies the printer driver version number assigned by the printer driver developer.
dmSize	Specifies the size, in bytes, of the DEVMODE structure except the dmDriverData (device-specific) member. If an application manipulates only the driver-independent portion of the data, it can use this member to determine the length of the structure without having to account for different versions.
dmDriverExtra	Contains the number of bytes of private driver-data that follow this structure. If a device driver does not use device-specific information, set this member to zero. See Remarks below.

dmFields	Specifies which of the remaining members in the DEVMODE structure have been initialized. Bit 0 (defined as DM_ORIENTATION) corresponds to dmOrientation; bit 1 (defined as DM_PAPERSIZE) specifies dmPaperSize, and so on. A printer driver supports only those members that are appropriate for the printer technology.
dmOrientation	Selects the orientation of the paper. This member can be set to either of the following:

Constant	Description
DMORIENT_PORTRAIT	
DMORIENT_LANDSCAPE	

dmPaperSize — Selects the size of the paper to print on. This member can be set to zero if the length and width of the paper are both set by the dmPaperLength and dmPaperWidth members. Otherwise, the dmPaperSize member can be set to one of the following predefined values:

Constant	Description
DMPAPER_LETTER	Letter, 8 1/2 by 11 inches
DMPAPER_LEGAL	Legal, 8 1/2 by 14 inches
DMPAPER_A4 A4	Sheet, 210 by 297 millimeters
DMPAPER_CSHEET	C Sheet, 17 by 22 inches
DMPAPER_DSHEET	D Sheet, 22 by 34 inches
DMPAPER_ESHEET	E Sheet, 34 by 44 inches
DMPAPER_LETTERSMALL	Letter Small, 8 1/2 by 11 inches
DMPAPER_TABLOID	Tabloid, 11 by 17 inches
DMPAPER_LEDGER	Ledger, 17 by 11 inches
DMPAPER_STATEMENT	Statement, 5 1/2 by 8 1/2 inches
DMPAPER_EXECUTIVE	Executive, 7 1/4 by 10 1/2 inches
DMPAPER_A3	A3 sheet, 297 by 420 millimeters
DMPAPER_A4SMALL	A4 small sheet, 210 by 297 millimeters
DMPAPER_A5	A5 sheet, 148 by 210 millimeters
DMPAPER_B4	B4 sheet, 250 by 354 millimeters
DMPAPER_B5	B5 sheet, 182-by-257-millimeter paper
DMPAPER_FOLIO	Folio, 8-1/2-by-13-inch paper
DMPAPER_QUARTO	Quarto, 215-by-275-millimeter paper
DMPAPER_10X14	10-by-14-inch sheet
DMPAPER_11X17	11-by-17-inch sheet
DMPAPER_NOTE	Note, 8 1/2 by 11 inches
DMPAPER_ENV_9	#9 Envelope, 3 7/8 by 8 7/8 inches
DMPAPER_ENV_10	#10 Envelope, 4 1/8 by 9 1/2 inches

	Constant	Description
	DMPAPER_ENV_11	#11 Envelope, 4 1/2 by 10 3/8 inches
	DMPAPER_ENV_12	#12 Envelope, 4 3/4 by 11 inches
	DMPAPER_ENV_14	#14 Envelope, 5 by 11 1/2 inches
	DMPAPER_ENV_DL	DL Envelope, 110 by 220 millimeters
	DMPAPER_ENV_C5	C5 Envelope, 162 by 229 millimeters
	DMPAPER_ENV_C3	C3 Envelope, 324 by 458 millimeters
	DMPAPER_ENV_C4	C4 Envelope, 229 by 324 millimeters
	DMPAPER_ENV_C6	C6 Envelope, 114 by 162 millimeters
	DMPAPER_ENV_C65	C65 Envelope, 114 by 229 millimeters
	DMPAPER_ENV_B4	B4 Envelope, 250 by 353 millimeters
	DMPAPER_ENV_B5	B5 Envelope, 176 by 250 millimeters
	DMPAPER_ENV_B6	B6 Envelope, 176 by 125 millimeters
	DMPAPER_ENV_ITALY	Italy Envelope, 110 by 230 millimeters
	DMPAPER_ENV_ MONARCH	Monarch Envelope, 3 7/8 by 7 1/2 inches
	DMPAPER_ENV_ PERSONAL	6 3/4 Envelope, 3 5/8 by 6 1/2 inches
	DMPAPER_FANFOLD_US	US Std Fanfold, 14 7/8 by 11 inches
	DMPAPER_FANFOLD_ STD_GERMAN	German Std Fanfold, 8 1/2 by 12 inches
	DMPA PER_FANFOLD_ LGL_GERMAN	German Legal Fanfold, 8 1/2 by 13 inches
dmPaperLength	Overrides the length of the paper specified by the dmPaperSize member, either for custom paper sizes or for devices such as dot-matrix printers, which can print on a page of arbitrary length. These values, along with all other values in this structure that specify a physical length, are in tenths of a millimeter.	
dmPaperWidth	Overrides the width of the paper specified by the dmPaperSize member.	
dmScale	Specifies the factor by which the printed output is to be scaled. The apparent page size is scaled from the physical page size by a factor of dmScale/100. For example, a letter-sized page with a dmScale value of 50 would contain as much data as a page of 17 by 22 inches because the output text and graphics would be half their original height and width.	

dmCopies	Selects the number of copies printed if the device supports multiple-page copies.
dmDefaultSource	Reserved. This member must be set to 0.
dmPrintQuality	Specifies the printer resolution. There are four predefined device-independent values. If a positive value is given, it specifies the number of dots per inch (DPI) and is therefore device dependent.

Constant	Description
DMRES_HIGH	
DMRES_MEDIUM	
DMlRES_LOW	
DMRES_DRAFT	

dmColor	Switches between color and monochrome on color printers. Following are the possible values:

Constant	Description
DMCOLOR_COLOR	
DMCOLOR_MONOCHROME	

dmDuplex	Selects duplex or double-sided printing for printers capable of duplex printing. Following are the possible values:

Constant	Description
DMDUP_SIMPLEX	
DMDUP_HORIZONTAL	
DMDUP_VERTICAL	

dmYResolution	Specifies the y-resolution, in dots per inch, of the printer. If the printer initializes this member, the dmPrintQuality member specifies the x-resolution, in dots per inch, of the printer.
dmTTOption	Specifies how TrueType® fonts should be printed. This member can be one of the following values:

Constant	Description
DMTT_BITMAP	Prints TrueType fonts as graphics. This is the default action for dot-matrix printers.
DMTT_DOWNLOAD	Downloads TrueType fonts as soft fonts. This is the default action for Hewlett-Packard printers that use Printer Control Language (PCL).
DMTT_SUBDEV	Substitute device fonts for TrueType fonts. This is the default action for PostScript® printers.

dmCollate	Specifies whether collation should be used when printing multiple copies. Using DMCOLLATE_FALSE provides faster, more efficient output, since the data is sent to a page printer just once, no matter how many copies are required. The printer is told to simply print the page again. This member can be be one of the following values:

Constant	Description
DMCOLLATE_TRUE	Collate when printing multiple copies.
DMCOLLATE_FALSE	Do NOT collate when printing multiple copies.

dmFormName	Specifies the name of the form to use; for example, Letter or Legal. A complete set of names can be retrieved through the Windows EnumForms function.
dmUnusedPadding	Used to align the structure to a DWORD boundary. This should not be used or referenced. Its name and usage is reserved, and can change in future releases.
dmBitsPerPel	Specifies in bits per pixel the color resolution of the display device. For example: 4 bits for 16 colors, 8 bits for 256 colors, or 16 bits for 65,536 colors.
dmPelsWidth	Specifies the width, in pixels, of the visible device surface.
dmPelsHeight	Specifies the height, in pixels, of the visible device surface.
dmDisplayFlags	Specifies the device's display mode. The following are valid flags:

Constant	Description
DM_GRAYSCALE	Specifies that the display is a non-color device. If this flag is not set, color is assumed.
DM_INTERLACED	Specifies that the display mode is interlaced. If the flag is not set, non-interlaced is assumed.

dmDisplayFrequency	Specifies the frequency, in hertz (cycles per second), of the display device in a particular mode.
dmICMMethod	Specifies how ICM is handled. For a non-ICM application, this member determines if ICM is enabled or disabled. For ICM applications, the system examines this member to determine how to handle ICM support. The printer driver must provide a user interface for setting this member. Most printer drivers support only the DMICMMETHOD_SYSTEM or DMICMMETHOD_NONE value. Drivers for PostScript printers support all values. This member can be one of the following predefined values, or a driver-defined value greater than or equal to the value of DMICMMETHOD_USER.

Constant	Description
DMICMMETHOD_NONE	Specifies that ICM is disabled.
DMICMMETHOD_SYSTEM	Specifies that ICM is handled by Windows.

	DMICMMETHOD_DRIVER	Specifies that ICM is handled by the device driver.
	DMICMMETHOD_DEVICE	Specifies that ICM is handled by the destination device.
dmICMIntent	Windows 95/98, Windows 2000: Specifies which of the three possible color matching methods, or intents, should be used by default. This member is primarily for non-ICM applications. ICM applications can establish intents by using the ICM functions. This member can be one of the following predefined values, or a driver defined value greater than or equal to the value of DMICM_USER.	
	Constant	**Description**
	DMICM_ABS_ COLORIMETRIC	Color matching should optimize to match the exact color requested without white point mapping. This value is most appropriate for use with proofing.
	DMICM_COLORMETRIC	Color matching should optimize to match the exact color requested. This value is most appropriate for use with business logos or other images when an exact color match is desired.
	DMICM_CONTRAST	Color matching should optimize for color contrast. This value is the most appropriate choice for scanned or photographic images when dithering is desired.
	DMICM_SATURATE	Color matching should optimize for color saturation. This value is the most appropriate choice for business graphs when dithering is not desired.
dmMediaType	Windows 95/98, Windows 2000: Specifies the type of media being printed on. The member can be one of the following predefined values, or a driver-defined value greater than or equal to the value of DMMEDIA_USER.	
	Constant	**Description**
	DMMEDIA_STANDARD	Plain paper.
	DMMEDIA_GLOSSY	Glossy paper.
	DMMEDIA_TRANSPARENCY	Transparent film.
dmDitherType	Windows 95/98, Windows 2000: Specifies how dithering is to be done. The member can be one of the following predefined values, or a driver-defined value greater than or equal to the value of DMDITHER_USER.	
	Constant	**Description**
	DMDITHER_NONE	No dithering.
	DMDITHER_COARSE	Dithering with a coarse brush.
	DMDITHER_FINE	Dithering with a fine brush.

	DMDITHER_LINEART	Line art dithering, a special dithering method that produces well defined borders between black, white, and gray scalings. It is not suitable for images that include continuous graduations in intensisty and hue such as scanned photographs.
	DMDITHER_ ERRORDIFFUSION	Windows 95/98: Dithering in which an algorithm is used to spread, or diffuse, the error of approximating a specified color over adjacent pixels. In contrast, DMDITHER_COARSE, DMDITHER_FINE, and DMDITHER_LINEART use patterned halftoning to approximate a color.
	DMDITHER_GRAYSCALE	Device does grayscaling.
dmReserved1	Windows 95/98, Windows 2000: Not used; must be zero.	
dmReserved2	Windows 95/98, Windows 2000: Not used; must be zero.	
dmPanningWidth	Windows NT/Windows 2000: This member must be zero. Windows 95/98: This member is not supported.	
dmPanningHeight	Windows NT/Windows 2000: This member must be zero. Windows 95/98: This member is not supported.	

Remarks

- Header file: Wingdi.h
- A device driver's private data follows the public portion of the DEVMODE structure. The size of the public data can vary for different versions of the structure. The dmSize member specifies the number of bytes of public data, and the dmDriverExtra member specifies the number of bytes of private data.
- Complete documentation for the DEVMODE Microsoft Windows API structures can be found at http://msdn.microsoft.com/

Print Engine Constants

The Print Engine constants are listed alphabetically in this section.

Area/Section Format Formula Constants

Constants PE_FFN_SECTION_VISIBILITY and PEP_FFN_SECTION_BACK_COLOUR are included for support of older applications. These constants has the same value as PE_FFN_AREASECTION_VISIBILITY and PEP_FFN_SECTION_BACK_COLOR. All new Crystal Reports applications should use PE_FFN_AREASECTION_VISIBILITY and PEP_FFN_SECTION_BACK_COLOR.

Constant	Description
PE_FFN_AREASECTION_VISIBILITY	Area and Section format
PE_FFN_SECTION_VISIBILITY	Section format
PE_FFN_SHOW_AREA	Area format
PE_FFN_NEW_PAGE_BEFORE	Area and Section format
PE_FFN_NEW_PAGE_AFTER	Area and Section format
PE_FFN_KEEP_TOGETHER	Area and Section format
PE_FFN_SUPPRESS_BLANK_SECTION	Section format
PE_FFN_RESET_PAGE_N_AFTER	Area and Section format
PE_FFN_PRINT_AT_BOTTOM_OF_PAGE	Area and Section format
PE_FFN_UNDERLAY_SECTION	Section format
PE_FFN_SECTION_BACK_COLOUR	Section format
PE_FFN_SECTION_BACK_COLOR	Section format

Chart Options Constants

The following chart options constants are supported.

“Chart Bar Size Constants” on page 542
“Chart Color Constants” on page 542
“Chart Data Point Constants” on page 542
“Chart Gridline Constants” on page 542
“Chart Legend Layout Constants” on page 543
“Chart Legend Placement Constants” on page 543
“Chart Marker Shape Constants” on page 543
“Chart Marker Size Constants” on page 543

Chart Bar Size Constants

Constant
PE_GBS_MINIMUMBARSIZE
PE_GBS_SMALLBARSIZE
PE_GBS_AVERAGEBARSIZE
PE_GBS_LARGEBARSIZE
PE_GBS_MAXIMUMBARSIZE

Chart Color Constants

Constant
PE_GCR_COLORCHART
PE_GCR_BLACKANDWHITECHART

Chart Data Point Constants

Constant
PE_GDP_NONE
PE_GDP_SHOWLABEL
PE_GDP_SHOWVALUE

Chart Gridline Constants

Constant
PE_GGT_NOGRIDLINES
PE_GGT_MINORGRIDLINES
PE_GGT_MAJORGRIDLINES
PE_GGT_MAJORANDMINORGRIDLINES

Chart Legend Layout Constants

Constant	Description
PE_GLL_PERCENTAGE	
PE_GLL_AMOUNT	
PE_GLL_CUSTOM	Used for PEGetGraphOptionInfo only. Do not use for PESetGraphOptionInfo.

Chart Legend Placement Constants

Constant
PE_GLP_PLACEUPPERRIGHT
PE_GLP_PLACEBOTTOMCENTER
PE_GLP_PLACETOPCENTER
PE_GLP_PLACERIGHT
PE_GLP_PLACELEFT

Chart Marker Shape Constants

Constant
PE_GMSP_RECTANGLESHAPE
PE_GMSP_CIRCLESHAPE
PE_GMSP_DIAMONDSHAPE
PE_GMSP_TRIANGLESHAPE

Chart Marker Size Constants

Constant
PE_GMS_SMALLMARKERS
PE_GMS_MEDIUMSMALLMARKERS
PE_GMS_MEDIUMMARKERS
PE_GMS_MEDIUMLARGEMARKERS
PE_GMS_LARGEMARKERS

Chart Number Format Constants

Constant
PE_GNF_NODECIMAL
PE_GNF_ONEDECIMAL
PE_GNF_TWODECIMAL
PE_GNF_CURRENCYNODECIMAL
PE_GNF_CURRENCYTWODECIMAL
PE_GNF_PERCENTNODECIMAL
PE_GNF_PERCENTONEDECIMAL
PE_GNF_PERCENTTWODECIMAL

Chart Pie Size Constants

Constant
PE_GPS_MINIMUMPIESIZE
PE_GPS_SMALLPIESIZE
PE_GPS_AVERAGEPIESIZE
PE_GPS_LARGEPIESIZE
PE_GPS_MAXIMUMPIESIZE

Chart Slice Detachment Constants

Constant
PE_GDPS_NODETACHMENT
PE_GDPS_SMALLESTSLICE
PE_GDPS_LARGESTSLICE

Chart Viewing Angle Constants

Constant
PE_GVA_STANDARDVIEW
PE_GVA_TALLVIEW
PE_GVA_TOPVIEW
PE_GVA_DISTORTEDVIEW
PE_GVA_SHORTVIEW
PE_GVA_GROUPEYEVIEW
PE_GVA_GROUPEMPHASISVIEW
PE_GVA_FEWSERIESVIEW
PE_GVA_FEWGROUPSVIEW
PE_GVA_DISTORTEDSTDVIEW
PE_GVA_THICKGROUPSVIEW
PE_GVA_SHORTERVIEW
PE_GVA_THICKSERIESVIEW
PE_GVA_THICKSTDVIEW
PE_GVA_BIRDSEYEVIEW
PE_GVA_MAXVIEW

Database Type Constants

Constant	Description
PE_DT_STANDARD	Standard, non-SQL databases.
PE_DT_SQL	SQL databases.
PE_DT_SQL_STORED_PROCEDURE	SQL stored procedures.

Error Codes

Constant	Value	Description
PE_ERR_NOERROR	0	
PE_ERR_NOTENOUGHMEMORY	500	
PE_ERR_INVALIDJOBNO	501	
PE_ERR_INVALIDHANDLE	502	
PE_ERR_STRINGTOOLONG	503	
PE_ERR_NOSUCHREPORT	504	

Constant	Value	Description
PE_ERR_NODESTINATION	505	
PE_ERR_BADFILENUMBER	506	
PE_ERR_BADFILENAME	507	
PE_ERR_BADFIELDNUMBER	508	
PE_ERR_BADFIELDNAME	509	
PE_ERR_BADFORMULANAME	510	
PE_ERR_BADSORTDIRECTION	511	
PE_ERR_ENGINENOTOPEN	512	
PE_ERR_INVALIDPRINTER	513	
PE_ERR_PRINTFILEEXISTS	514	
PE_ERR_BADFORMULATEXT	515	
PE_ERR_BADGROUPSECTION	516	
PE_ERR_ENGINEBUSY	517	
PE_ERR_BADSECTION	518	
PE_ERR_NOPRINTWINDOW	519	
PE_ERR_JOBALREADYSTARTED	520	
PE_ERR_BADSUMMARYFIELD	521	
PE_ERR_NOTENOUGHSYSRES	522	
PE_ERR_BADGROUPCONDITION	523	
PE_ERR_JOBBUSY	524	
PE_ERR_BADREPORTFILE	525	
PE_ERR_NODEFAULTPRINTER	526	
PE_ERR_SQLSERVERERROR	527	
PE_ERR_BADLINENUMBER	528	
PE_ERR_DISKFULL	529	
PE_ERR_FILEERROR	530	
PE_ERR_INCORRECTPASSWORD	531	
PE_ERR_BADDATABASEDLL	532	
PE_ERR_BADDATABASEFILE	533	
PE_ERR_ERRORINDATABASEDLL	534	
PE_ERR_DATABASESESSION	535	
PE_ERR_DATABASELOGON	536	
PE_ERR_DATABASELOCATION	537	
PE_ERR_BADSTRUCTSIZE	538	

Constant	Value	Description
PE_ERR_BADDATE	539	
PE_ERR_BADEXPORTDLL	540	
PE_ERR_ERRORINEXPORTDLL	541	
PE_ERR_PREVATFIRSTPAGE	542	
PE_ERR_NEXTATLASTPAGE	543	
PE_ERR_CANNOTACCESSREPORT	544	
PE_ERR_USERCANCELLED	545	
PE_ERR_OLE2NOTLOADED	546	
PE_ERR_BADCROSSTABGROUP	547	
PE_ERR_NOCTSUMMARIZEDFIELD	548	
PE_ERR_DESTINATIONNOTEXPORT	549	
PE_ERR_INVALIDPAGENUMBER	550	
PE_ERR_NOTSTOREDPROCEDURE	552	
PE_ERR_INVALIDPARAMETER	553	
PE_ERR_GRAPHNOTFOUND	554	
PE_ERR_INVALIDGRAPHTYPE	555	
PE_ERR_INVALIDGRAPHDATA	556	
PE_ERR_CANNOTMOVEGRAPH	557	
PE_ERR_INVALIDGRAPHTEXT	558	
PE_ERR_INVALIDGRAPHOPT	559	
PE_ERR_BADSECTIONHEIGHT	560	
PE_ERR_BADVALUETYPE	561	
PE_ERR_INVALIDSUBREPORTNAME	562	
PE_ERR_NOPARENTWINDOW	564	Dialog parent window
PE_ERR_INVALIDZOOMFACTOR	565	Zoom factor
PE_ERR_PAGESIZEOVERFLOW	567	
PE_ERR_LOWSYSTEMRESOURCES	568	
PE_ERR_INVALIDOBJECTFORMATNAME	571	
PE_ERR_INVALIDNEGATIVEVALUE	572	
PE_ERR_INVALIDMEMORYPOINTER	573	
PE_ERR_INVALIDOBJECTTYPE	574	
PE_ERR_INVALIDGRAPHDATATYPE	577	
PE_ERR_INVALIDSUBREPORTLINKNUMBER	582	
PE_ERR_SUBREPORTLINKEXIST	583	

Constant	Value	Description
PE_ERR_BADROWCOLVALUE	584	
PE_ERR_INVALIDSUMMARYNUMBER	585	
PE_ERR_INVALIDGRAPHDATAFIELDNUMBER	586	
PE_ERR_INVALIDSUBREPORTNUMBER	587	
PE_ERR_INVALIDFIELDSCOPE	588	
PE_ERR_FIELDINUSE	590	
PE_ERR_INVALIDPARAMETERNUMBER	594	
PE_ERR_INVALIDPAGEMARGINS	595	
PE_ERR_REPORTONSECUREQUERY	596	
PE_ERR_CANNOTOPENSECUREQUERY	597	
PE_ERR_INVALIDSECTIONNUMBER	598	
PE_ERR_SQLSERVERNOTOPENED	599	
PE_ERR_TABLENAMEEXIST	606	
PE_ERR_INVALIDCURSOR	607	
PE_ERR_FIRSTPASSNOTFINISHED	608	
PE_ERR_CREATEDATASOURCE	609	
PE_ERR_CREATEDRILLDOWNPARAMETERS	610	
PE_ERR_CHECKFORDATASOURCECHANGES	613	
PE_ERR_STARTBACKGROUNDPROCESSING	614	
PE_ERR_SQLSERVERINUSE	619	
PE_ERR_GROUPSORTFIELDNOTSET	620	
PE_ERR_CANNOTSETSAVESUMMARIES	621	
PE_ERR_LOADOLAPDATABASEMANAGER	622	
PE_ERR_OPENOLAPCUBE	623	
PE_ERR_READOLAPCUBEDATA	624	
PE_ERR_CANNOTSAVEQUERY	626	
PE_ERR_CANNOTREADQUERYDATA	627	
PE_ERR_MAINREPORTFIELDLINKED	629	
PE_ERR_INVALIDMAPPINGTYPEVALUE	630	
PE_ERR_HITTESTFAILED	636	
PE_ERR_BADSQLEXPRESSIONNAME	637	No SQL expression by the specified *name* exists in this report.
PE_ERR_BADSQLEXPRESSIONNUMBER	638	No SQL expression by the specified *number* exists in this report.
PE_ERR_BADSQLEXPRESSIONTEXT	639	Not a valid SQL expression

Constant	Value	Description
PE_ERR_INVALIDDEFAULTVALUEINDEX	641	Invalid index for default value of a parameter.
PE_ERR_NOMINMAXVALUE	642	The specified PE_PF_* type does not have min / max values.
PE_ERR_INCONSISTANTTYPES	643	If both min and max values are specified in PESetParameterMinMaxValue, the value types for the min and max must be the same.
PE_ERR_CANNOTLINKTABLES	645	
PE_ERR_CREATEROUTER	646	
PE_ERR_INVALIDFIELDINDEX	647	
PE_ERR_INVALIDGRAPHTITLETYPE	648	
PE_ERR_INVALIDGRAPHTITLEFONTTYPE	649	
PE_ERR_PARAMTYPEDIFFERENT	650	The type used in a add / set value API for a parameter differs with it's existing type.
PE_ERR_INCONSISTANTRANGETYPES	651	The value type for both start & end range values must be the same.
PE_ERR_RANGEORDISCRETE	652	An operation was attempted on a discrete parameter that is only legal for range parameters or vice versa.
PE_ERR_NOTMAINREPORT	654	An operation was attempted that is disallowed for subreports.
PE_ERR_INVALIDCURRENTVALUEINDEX	655	Invalid index for current value of a parameter.
PE_ERR_LINKEDPARAMVALUE	656	Operation illegal on linked parameter.
PE_ERR_INVALIDPARAMETERRANGEINFO	672	Invalid PE_RI_XXX combination.
PE_ERR_INVALIDSORTMETHODINDEX	674	Invalid sort method index.
PE_ERR_INVALIDGRAPHSUBTYPE	675	Invalid PE_GST_XXX or PE_GST_XXX does not match PE_GT_XXX or PE_GST_XXX current graph type.
PE_ERR_BADGRAPHOPTIONINFO	676	One of the members of PEGraphOptionInfo is out of range.
PE_ERR_BADGRAPHAXISINFO	677	One of the members of PEGraphAxisInfo is out of range.
PE_ERR_INVALIDPARAMETERVALUE	687	

Constant	Value	Description
PE_ERR_INVALIDFORMULASYNTAXTYPE	688	Specified formula syntax not in PE_FST_XXX
PE_ERR_INVALIDCROPVALUE	689	
PE_ERR_INVALIDCOLLATIONVALUE	690	
PE_ERR_STARTPAGEGREATERSTOPPAGE	691	
PE_ERR_READONLYPARAMETEROPTION	700	
PE_ERR_MINGREATERTHANMAX	701	
PE_ERR_INVALIDSTARTPAGE	702	
PE_ERR_OTHERERROR	997	
PE_ERR_INTERNALERROR	998	Programming error
PE_ERR_NOTIMPLEMENTED	999	

Event Id Constants

Constant	Description
PE_CLOSE_PRINT_WINDOW_EVENT	
PE_ACTIVATE_PRINT_WINDOW_EVENT	
PE_DEACTIVATE_PRINT_WINDOW_EVENT	
PE_PRINT_BUTTON_CLICKED_EVENT	
PE_EXPORT_BUTTON_CLICKED_EVENT	
PE_ZOOM_LEVEL_CHANGING_EVENT	
PE_FIRST_PAGE_BUTTON_CLICKED_EVENT	
PE_PREVIOUS_PAGE_BUTTON_CLICKED_EVENT	
PE_NEXT_PAGE_BUTTON_CLICKED_EVENT	
PE_LAST_PAGE_BUTTON_CLICKED_EVENT	
PE_CANCEL_BUTTON_CLICKED_EVENT	
PE_CLOSE_BUTTON_CLICKED_EVENT	
PE_SEARCH_BUTTON_CLICKED_EVENT	
PE_GROUP_TREE_BUTTON_CLICKED_EVENT	
PE_PRINT_SETUP_BUTTON_CLICKED_EVENT	
PE_REFRESH_BUTTON_CLICKED_EVENT	
PE_SHOW_GROUP_EVENT	
PE_DRILL_ON_GROUP_EVENT	Include drill on graph.
PE_DRILL_ON_DETAIL_EVENT	
PE_READING_RECORDS_EVENT	

Constant	Description
PE_START_EVENT	
PE_STOP_EVENT	
PE_MAPPING_FIELD_EVENT	
PE_RIGHT_CLICK_EVENT	Right mouse click.
PE_LEFT_CLICK_EVENT	Left mouse click.
PE_MIDDLE_CLICK_EVENT	Middle mouse click.
PE_DRILL_ON_HYPERLINK_EVENT	
PE_LAUNCH_SEAGATE_ANALYSIS_EVENT	

Field Mapping Type Constants

Constant	Description
PE_FM_AUTO_FLD_MAP	Automatic field name mapping.
PE_FM_CRPE_PROMPT_FLD_MAP	CRPE provides dialog box to map field manually.
PE_FM_EVENT_DEFINED_FLD_MAP	CRPE provides list of fields in the report and the new database. This constant is not available in global32.bas.

Formula Syntax Constants

Constant	Description
PE_FST_CRYSTAL	Default value. See Remarks below.
PE_FST_BASIC	See Remarks below.

Remarks

For Running Total Condition Formulas:

- PE_FST_CRYSTAL is the syntax for the eval formula.
- PE_FST_BASIC is the syntax for the reset formula.

Graph Subtype Constants

The following chart subtypes are supported.

“Bar Charts” on page 552
“Line Charts” on page 552
“Area Charts” on page 552
“Pie Charts” on page 553
“Doughnut Charts” on page 553
“3D Riser Charts” on page 553

Bar Charts

Constant
PE_GST_SIDEBYSIDEBARCHART
PE_GST_STACKEDBARCHART
PE_GST_PERCENTBARCHART
PE_GST_FAKED3DSIDEBYSIDEBARCHART
PE_GST_FAKED3DSTACKEDBARCHART
PE_GST_FAKED3DPERCENTBARCHART

Line Charts

Constant
PE_GST_REGULARLINECHART
PE_GST_STACKEDLINECHART
PE_GST_PERCENTAGELINECHART
PE_GST_LINECHARTWITHMARKERS
PE_GST_STACKEDLINECHARTWITHMARKERS
PE_GST_PERCENTAGELINECHARTWITHMARKERS

Area Charts

Constant
PE_GST_ABSOLUTEAREACHART
PE_GST_STACKEDAREACHART
PE_GST_PERCENTAREACHART
PE_GST_FAKED3DABSOLUTEAREACHART
PE_GST_FAKED3DSTACKEDAREACHART
PE_GST_FAKED3DPERCENTAREACHART

Pie Charts

Constant
PE_GST_REGULARPIECHART
PE_GST_FAKED3DREGULARPIECHART
PE_GST_MULTIPLEPIECHART
PE_GST_MULTIPLEPROPORTIONALPIECHART

Doughnut Charts

Constant
PE_GST_REGULARDOUGHNUTCHART
PE_GST_MULTIPLEDOUGHNUTCHART
PE_GST_MULTIPLEPROPORTIONALDOUGHNUTCHART

3D Riser Charts

Constant
PE_GST_THREEDREGULARCHART
PE_GST_THREEDPYRAMIDCHART
PE_GST_THREEDOCTAGONCHART
PE_GST_THREEDCUTCORNERSCHART

3D Surface Charts

Constant
PE_GST_THREEDSURFACEREGULARCHART
PE_GST_THREEDSURFACEWITHSIDESCHART
PE_GST_THREEDSURFACEHONEYCOMBCHART

Scatter Charts

Constant
PE_GST_XYSCATTERCHART
PE_GST_XYSCATTERDUALAXISCHART
PE_GST_XYSCATTERWITHLABELSCHART
PE_GST_XYSCATTERDUALAXISWITHLABELSCHART

Radar Charts

Constant
PE_GST_REGULARRADARCHART
PE_GST_STACKEDRADARCHART
PE_GST_RADARDUALAXISCHART

Bubble Charts

Constant
PE_GST_REGULARBUBBLECHART
PE_GST_DUALAXISBUBBLECHART

Stock (High/Low/Close Type) Charts

Constant
PE_GST_HIGHLOWCHART
PE_GST_HIGHLOWOPENCLOSECHART

Misc Chart Types

Constant
PE_GST_UNKNOWNSUBTYPECHART

Graph Text Font Constants

Constant
PE_GTF_TITLEFONT
PE_GTF_SUBTITLEFONT
PE_GTF_FOOTNOTEFONT
PE_GTF_GROUPSTITLEFONT
PE_GTF_DATATITLEFONT
PE_GTF_LEGENDFONT
PE_GTF_GROUPLABELSFONT
PE_GTF_DATALABELSFONT

Graph Title Type Constants

Constant
PE_GTT_TITLE
PE_GTT_SUBTITLE
PE_GTT_FOOTNOTE
PE_GTT_SERIESTITLE
PE_GTT_GROUPSTITLE
PE_GTT_XAXISTITLE
PE_GTT_YAXISTITLE
PE_GTT_ZAXISTITLE

Graph Type Constants

Constant	Description
PE_GT_BARCHART	
PE_GT_LINECHART	
PE_GT_AREACHART	
PE_GT_PIECHART	
PE_GT_DOUGHNUTCHART	
PE_GT_THREEDRISERCHART	
PE_GT_THREEDSURFACECHART	
PE_GT_SCATTERCHART	
PE_GT_RADARCHART	
PE_GT_BUBBLECHART	
PE_GT_STOCKCHART	
PE_GT_USERDEFINEDCHART	Use for PEGetGraphTypeInfo only. Do not use for PESetGraphTypeInfo.
PE_GT_UNKNOWNTYPECHART	Use for PEGetGraphTypeInfo only. Do not use for PESetGraphTypeInfo.

Group Condition Constants

The following tables list the Group Condition masks and constants applicable to various field types.

Group Condition Masks and Type Constants

Mask and Type Constants	Value
PE_GC_CONDITIONMASK	0x00FF
PE_GC_TYPEMASK	0x0F00
PE_GC_TYPEOTHER	0x0000
PE_GC_TYPEDATE	0x0200
PE_GC_TYPEBOOLEAN	0x0400
PE_GC_TYPETIME	0x0800

All field types except Date and Boolean

Constant	Description
PE_GC_ANYCHANGE	Triggers a grouping every time there is a change.

Date and DateTime Fields

Constant (Date Fields Only)	Description
PE_GC_DAILY	Triggers a grouping every time the date changes.
PE_GC_WEEKLY	Triggers a grouping every time the date changes from one week to the next. (A week runs from Sunday through Saturday).
PE_GC_BIWEEKLY	Triggers a grouping every time the date changes from one two-week period to the next. (A week runs from Sunday through Saturday).
PE_GC_SEMIMONTHLY	Triggers a grouping every time the date changes from one half-month period to the next.
PE_GC_MONTHLY	Triggers a grouping every time the date changes from one month to the next.
PE_GC_QUARTERLY	Triggers a grouping every time the date changes from one calendar quarter to the next.

Constant (Date Fields Only)	Description
PE_GC_SEMIANNUALLY	Triggers a grouping every time the date changes from one half-year period to the next.
PE_GC_ANNUALLY	Triggers a grouping every time the date changes from one year to the next.

DateTime and Time Fields

Constant
PE_GC_BYSECOND
PE_GC_BYMINUTE
PE_GC_BYHOUR
PE_GC_BYAMPM

Boolean Fields

Constant (Boolean Fields Only)	Description
PE_GC_TOYES	Triggers a grouping every time the sort- and group-by field value changes from No to Yes.
PE_GC_TONO	Triggers a grouping every time the sort- and group-by field value changes from Yes to No.
PE_GC_EVERYYES	Triggers a grouping every time the sort- and group-by field value is Yes.
PE_GC_EVERYNO	Triggers a grouping every time the sort- and group-by field value is No.
PE_GC_NEXTISYES	Triggers a grouping every time the next value in the sort- and group-by field is Yes.
PE_GC_NEXTISNO	Triggers a grouping every time the next value in the sort- and group-by field is No.

Job Destination Constants

Constant	Description
PE_TO_NOWHERE	No destination.
PE_TO_WINDOW	Print to window.
PE_TO_PRINTER	Print to printer.
PE_TO_EXPORT	Export.
PE_FROM_QUERY	From a query.

Job Status Constants

Constant	Description
PE_JOBNOTSTARTED	
PE_JOBINPROGRESS	
PE_JOBCOMPLETED	
PE_JOBFAILED	An error occurred.
PE_JOBCANCELLED	The job was canceled by user.
PE_JOBHALTED	The job was halted because of too many records or too much time.

Object Type Constants

Constant
PE_OI_FIELDOBJECT
PE_OI_TEXTOBJECT
PE_OI_LINEOBJECT
PE_OI_BOXOBJECT
PE_OI_SUBREPORTOBJECT
PE_OI_OLEOBJECT
PE_OI_GRAPHOBJECT
PE_OI_CROSSTABOBJECT
PE_OI_BLOBFIELDOBJECT
PE_OI_MAPOBJECT
PE_OI_OLAPGRIDOBJECT

Ole Object Type Constants

Constant
PE_OOI_LINKEDOBJECT
PE_OOI_EMBEDDEDOBJECT
PE_OOI_STATICOBJECT

Ole Object Update Constants

Constant
PE_OOI_AUTOUPDATE
PE_OOI_MANUALUPDATE

Parameter Field Value Type Constants

Constant
PE_PF_NUMBER
PE_PF_CURRENCY
PE_PF_BOOLEAN
PE_PF_DATE
PE_PF_STRING
PE_PF_DATETIME
PE_PF_TIME

Range Info Constants

Constant	Value
PE_RI_INCLUDEUPPERBOUND	1
PE_RI_INCLUDELOWERBOUND	2
PE_RI_NOUPPERBOUND	4
PE_RI_NOLOWERBOUND	8

Section Codes

Constant
PE_ALLSECTIONS
PE_SECT_PAGE_HEADER
PE_SECT_PAGE_FOOTER
PE_SECT_REPORT_HEADER
PE_SECT_REPORT_FOOTER
PE_SECT_GROUP_HEADER
PE_SECT_GROUP_FOOTER
PE_SECT_DETAIL

Sort Method Constants

Constant
PE_OR_NO_SORT
PE_OR_ALPHANUMERIC_ASCENDING
PE_OR_ALPHANUMERIC_DESCENDING
PE_OR_NUMERIC_ASCENDING
PE_OR_NUMERIC_DESCENDING

Sort Order Constants

Constant	Description
PE_SF_DESCENDING	Sorts data in descending order (Z to A, 9 to 1).
PE_SF_ASCENDING	Sorts data in ascending order (A to Z, 1 to 9).
PE_SF_ORIGINAL	Group condition only: Sorts data in its original order.
PE_SF_SPECIFIED	Group condition only: Sorts data in a specified order. Read only.

Track Cursor Constants

Constant	Description
PE_TC_DEFAULT_CURSOR	CRPE default cursor = PE_TC_ARROW_CURSOR.
PE_TC_ARROW_CURSOR	Arrow cursor.
PE_TC_CROSS_CURSOR	Cross cursor.
PE_TC_IBEAM_CURSOR	I-beam cursor.
PE_TC_UPARROW_CURSOR	Arrow cursor pointing to the top of the screen.
PE_TC_SIZEALL_CURSOR	32-bit only.
PE_TC_SIZENWSE_CURSOR	Sizing cursor when resizing from the top, left-hand side of the screen to the bottom, right-hand side of the screen.
PE_TC_SIZENESW_CURSOR	Sizing cursor when resizing from the top, right-hand side of the screen to the bottom, left-hand side of the screen.
PE_TC_SIZEWE_CURSOR	Sizing cursor when resizing from the left side of the screen to the right side of the screen.
PE_TC_SIZENS_CURSOR	Sizing cursor when resizing from the top of the screen to the bottom of the screen.
PE_TC_NO_CURSOR	32-bit only.

Constant	Description
PE_TC_WAIT_CURSOR	Wait (i.e., hourglass) cursor.
PE_TC_APPSTARTING_CURSOR	32-bit only.
PE_TC_HELP_CURSOR	32-bit only.
PE_TC_SIZE_CURSOR	Not used in 32-bit applications. Use PE_TC_SIZEALL_CURSOR.
PE_TC_ICON_CURSOR	Not used in 32-bit applications. Use PE_TC_ARROW_CURSOR.
PE_TC_BACKGROUND_PROCESS_CURSOR	CRPE specific cursor.
PE_TC_GRAB_HAND_CURSOR	CRPE specific cursor.
PE_TC_ZOOM_IN_CURSOR	CRPE specific cursor.
PE_TC_REPORT_SECTION_CURSOR	CRPE specific cursor.
PE_TC_HAND_CURSOR	CRPE specific cursor.
PE_TC_MAGNIFY_CURSOR	CRPE specific cursor. Magnifying glass cursor (used for drill-down).

Zoom Level Constants

Constant
PE_ZOOM_FULL_SIZE
PE_ZOOM_SIZE_FIT_ONE_SIDE
PE_ZOOM_SIZE_FIT_BOTH_SIDES

Obsolete Functions, Structures, and Constants

The following obsolete functions, structures, and constants are listed alphabetically. They are not supported by the current version. Where appropriate, replacement or updated equivalents are listed. The corresponding new call should be used in all new applications.

Obsolete Functions

Obsolete Function	Replacement Function
PEGetGraphData	
PEGetGraphOptions	“PEGetGraphOptionInfo” on page 311
PEGetGraphText	“PEGetGraphTextInfo” on page 312
PEGetGraphType	“PEGetGraphTypeInfo” on page 313

Obsolete Function	Replacement Function
PEGetMinimumSectionHeight	“PEGetSectionHeight” on page 363
PEGetNParams	“PEGetNParameterFields” on page 327
PEGetNthParam	“PEGetNthParameterField” on page 340
PEGetNthParamInfo	“PEGetParameterValueInfo” on page 354
PESetGraphData	
PESetGraphOptions	“PESetGroupOptions” on page 414
PESetGraphText	“PESetGraphTextInfo” on page 411
PESetGraphType	“PESetGraphTypeInfo” on page 412
PESetMinimumSectionHeight	“PESetSectionHeight” on page 441
PESetNthParam	“PESetNthParameterField” on page 423

Obsolete Structures

Obsolete Structure
PECharSepFileOptions
PEGraphDataInfo
PEGraphOptions
PEGraphTextInfo
PEParameterInfo
PEPrintFileOptions

Obsolete Constants

Obsolete Constants
PE_GRAPH_XXX Graph Direction Constants
PE_SIDE_ / PE_FAKED_ / etc. Graph Type Constants

Active Data 7

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

Active Data Driver

Modern Visual Basic applications often use advanced ActiveX components to connect to data sources. These data sources may include Data Access Objects (DAO), Remote Data Objects (RDO), OLE DB providers, such as ActiveX Data Objects (ADO), or the Visual Basic data controls. Using the Active Data Driver for Crystal Reports, you can design reports for your Visual Basic applications that use these same ActiveX data sources. The Active Data Driver also supports Crystal Data Objects (CDO) and the Crystal Data Source Type Library. 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 575. For information on the Crystal Data Source Type Library, see “Crystal Data Source Type Library” on page 579.

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 Active Data Driver supports the use of Data definition files, 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, of course, 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 Crystal Active Data 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:

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. For an example of a data definition file, refer to the file ORDERS.TTX installed in the \Program Files\Seagate Software\Crystal Reports directory.

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 Crystal Active Data 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 575), and the Crystal Data Source Type Library (see "Crystal Data Source Type Library" on page 579).

Once the Crystal Active Data Driver obtains the Recordset from the runtime data source, the Report Designer Component can generate the actual report using 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 569.

Using the Active Data Driver

Designing and generating reports using the Crystal Active Data Driver is a straightforward process, but requires several specific steps:

- "Select the design time data source" on page 566
- "Design the Report" on page 566
- "Obtain a Recordset from the Runtime Data Source" on page 567
- "Open the Report" on page 568
- "Pass the Recordset to the Active Data Driver" on page 568
- "Print the Report" on page 569

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 Driver, or you can specify a data definition file so that the actual data is specified at runtime only. The following example uses the sample data definition file included with Crystal Reports:

1. Click **New Report** in the Crystal Reports Welcome dialog box, or click the **New** button on the Crystal Reports toolbar.
2. In the Crystal Report Gallery dialog box click **Using the Report Expert**. In this example, you can click **Standard** from the Choose an Expert box. Then Click **OK**.
3. In the Standard Report Expert Dialog box click **Database**.
4. In the Data Explorer dialog box:
   - expand **More Data Sources**
   - expand **Active Data**
   - expand **Active Data (ADO)**.
5. In the Select Data Source dialog box click the ODBC (ADO) option, select Xtreme Sample Database from the drop-down list, and then click OK.
6. In the Data Explorer dialog box click **Add**.
7. In the Select Recordset dialog box select **Orders** from the Object list, and then click **OK**.
8. In the Data Explorer dialog box click **Close**. The Orders table appears as *ado* in the Tables available for report box, under the data tab, in the Standard Report Expert dialog 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 572.

Design the Report

Once you have selected a data definition file or an ActiveX data source, you can design your report just as you would design any other report.

1. Click the **Fields** Tab of the Standard Report Expert.
   The data definition file *orders* appears as a database table in the Database Fields list box. Each of the fields defined in orders.ttx appears as a field in the orders table.
2. Add fields to your report just as you would normally add fields to a report using the Standard Report Expert.
3. Continue designing the report using the Standard Report Expert. When finished, click **Design Report**. Since the report is based on a data definition file, there is no point in previewing it at this time.
4. Apply any formatting or other changes that you feel are necessary to fine-tune the look of your report. Save the report when finished.

Note: Before saving your report, be sure to turn off the Save Data with Report option under 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 Crystal Active Data Driver in conjunction with the Report Designer Component or one of the other Crystal Reports development tools. See the "Visual Basic Solutions" chapter in the *Crystal Reports Technical Reference Guide*, or the *Crystal Reports Developer's Help (CrystalDevHelp.chm)* for more information on the other Crystal Reports development tools.

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 65.

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 575. For information on the Crystal Data Source Type Library, see "Crystal Data Source Type Library" on page 579.

This tutorial creates a Recordset object from the Orders table of the XTREME.MDB sample database using DAO. 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 575).

Note: You must add the Data Access Objects component to your Visual Basic project before performing the following steps. For instructions on using DAO 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 db As New DAO.Database
Dim rs As DAO.Recordset
```

- Obtain a Database object from the Xtreme database.

```
Set db = DBEngine.Workspaces(0).OpenDatabase( _
"c:\Program Files\Seagate Software\Crystal Reports\xtreme.mdb")
```

- Obtain a Recordset object from the Orders table of the Xtreme database.

```
Set rs = db.OpenRecordset("Orders", dbOpenTable)
```

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. For complete information on using the Automation Server, see "Crystal Report Engine Automation Server" on page 12.

- 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 ORDERS.RPT.

```
Set CRXReport = CRXApplication.OpenReport("c:\reports\Orders.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 Report Designer Component Automation Server uses the Active Data Driver itself to replace the data definition file, at runtime, with the 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 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 DAO 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 65 for more information.

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, etc.) 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.

For complete information on creating data definition files, see “Creating Data Definition Files” on page 569. Crystal Reports installs a sample data definition file in the \Program Files\Seagate Software\Crystal Reports directory on your system. This file is named ORDERS.TTX and can be used with the Orders table in the XTREME.MDB sample database or the Xtreme sample data ODBC data source that was created when you installed Crystal Reports.

The following is an example of how fields are defined in a data definition file:

```
Order ID    Long    1
Customer NameString50Sample string value
Order Date  Date    Jan 5, 2000
Order AmountCurrency$1.00
```

The Active Data 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: ■ Jan 5, 1999 ■ 07 / 11 / 97 5:06:07 ■ 07 / 11 / 97 ■ 23:30:01
Long, int32	32-bit integer value.
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 tools for simplifying the process. Each tool has its advantages. Review the process for using each tool described below to determine which best suits your own environment and development process.

Database Definition Tool

The Database Definition Tool is available from the Select Data Source dialog box when you begin designing a report based on the Active Data Driver. This tool allows you to design a data definition file as the first step of designing your report. From the Standard Report Expert:

1. In the Standard Report Expert click **Database**.
2. In the Data Explorer dialog box:
   - expand **More Data Sources**
   - expand **Active Data**
   - expand **Active Data (Field Definitions Only)**.
3. In the Select Data Source dialog box click **New** to create a new data definition file.

   The Database Definition Tool appears.
4. 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.
5. 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.
6. 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.
7. You can delete a field that you have created by selecting the field in the list box and clicking **Delete**.
8. Click the **Close** button in the upper right of the Database Definition Tool dialog box when you are finished designing your data definition file. A message appears asking if you want to save the data definition file.
9. Click **Yes**, and a Save File As dialog box appears.
10. Save the data definition file where it can be accessed by your report file. When finished, the new data definition file will appear in the Data Definition text box in the Select Data Source dialog box.
11. Continue creating your report.

Active Data Driver Functions

The Active Data Driver (P2SMON.DLL) is a standard dynamic link library that is normally used by Crystal Reports (or the Report Designer Component) to access ActiveX data sources such as DAO and ADO. The DLL is installed, by default, in your \WINDOWS\SYSTEM directory. In addition, the Active Data Driver exports functions that can be used at runtime from within your application to dynamically design a data definition file based on your data source, and a report file based on the data definition file. These functions are available to any development environment that supports DLL function calls.

Note: To use the functions in the Active Data Driver DLL, you must declare the functions first. Refer to your Visual Basic documentation for information on declaring DLL functions. See "The Crystal Active Data Driver Reference" on page 601 for information about declaring the Active Data Driver functions.

To use the Active Data Driver Functions from Visual Basic

1. Obtain a valid Recordset object from your DAO, ADO, or Data Control data source, or a valid Rowset object using CDO.
2. Call the function CreateReportOnRuntimeDS to create a data definition file based on your Recordset or Rowset object. For example:

```
CreateReportOnRuntimeDS(daoRs, "c:\reports\orders.rpt",
    "c:\reports\orders.ttx", True, False)
```

This example creates a data definition file named ORDERS.TTX, then creates a simple report file based on this data definition file and names it ORDERS.RPT. If the last argument is set to True, Crystal Reports, if installed on the system, will open automatically on the user's machine, allowing them to make changes to the report file.

Notice that the first argument is a DAO Recordset object. If you are using this function in a language such as C or C++, you would pass a pointer to an IUnknown derived interface to the Recordset.

Note: See "The Crystal Active Data Driver Reference" on page 601 for complete information on the functions provided by the Active Data Driver.

Using ActiveX Data Sources at Design Time

The Active Data Driver is 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 simply specify an ADO data source at design time for the report.

The Select Data Source dialog opens when you select one of the active data folders in the More Data Sources \ Active Data folder in the Data Explorer. This dialog box provides four options for selecting a data source to use in your report: specify an ODBC data source for ADO or RDO, specify an ADO connection string for OLE DB, specify a DAO recordset, 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, or DAO data source.

The following topics are discussed in this section:

ODBC with ADO and RDO

1. Click the ODBC option in the Select Data Source dialog box.

   This option allows you to connect to an ODBC data source through ADO or RDO. The currently selected data objects technology appears in parentheses next to the ODBC option.

- Use the drop-down list to select an ODBC data source that is available on your system.
- Click the **New** button to create a new ODBC data source. Refer to Microsoft ODBC documentation for information on creating ODBC data sources.
- Click the **Advanced** button to select ADO or RDO as the data objects technology used. This should match the technology used in your Visual Basic application.

2. After you select your data source and data objects technology, you can click **Next** in the Select Data Source dialog box.

   The Select Recordset dialog box appears.

3. If the ODBC data source requires log on information, specify a user name and password to log on.

4. Determine if you want to create a Recordset or Resultset using an object available from your data source, such as a database table, or if you prefer to specify a SQL statement. Select the appropriate option in the Recordset section of the Select Recordset dialog box.

   Note: For simplicity, RDO Resultsets are also referred to as Recordsets in this dialog box.

5. If you want to connect to a database object, use the Object Type drop-down list box to select the type of database object, such as a Table, then select the object itself from the Object drop-down list box.

6. If you want to obtain a Recordset using a SQL statement, write the SQL statement in the text box provided, or click Build to use the Microsoft Query application and Query Wizard to visually design your SQL statement.

7. Click **Finish** in the Select Recordset dialog box. Either *ado* or *rdo* will appear in the list box on the Data Tab of the Standard Report Expert.

8. Continue creating your report normally. While creating your report, the *ado* or *rdo* specification will act like a database table, providing all fields that have been obtained from your ADO Recordset or RDO Resultset.

ADO and OLE DB

1. Click the ADO and OLE DB option in the Select Data Source dialog box. This option is designed to allow you to specify an ADO connection string that can connect to any OLE DB provider.

2. Type the ADO connection string into the text box provided, or click Build to open the Datal Links Properties dialog box. The following are examples of an acceptable connection string for ADO:

   DSN=Xtreme sample data;

   DATABASE=pubs;DSN=Publishers;UID=sa;Password=;

3. Type in the first example shown here to follow along in this tutorial. Click **Next** in the Select Data Source dialog box when finished. The Select Recordset dialog box appears.

4. Determine if you want to create a Recordset using an object available from your data source, such as a database table, or if you prefer to specify a SQL statement. Select the appropriate option in the Recordset section of the Select Recordset dialog box.

5. If you want to connect to a database object, use the Object Type drop-down list to select the type of database object, such as a Table, then select the object itself from the Object drop-down list.

6. If you want to obtain a Recordset using a SQL statement, write the SQL statement in the text box provided, or click Build to use the Microsoft Query application and Query Wizard to visually design your SQL statement.

7. Click **Finish** in the Select Recordset dialog box. You will see *ado* in the list box on the Data Tab of the Standard Report Expert.

8. Continue creating your report normally. While creating your report, the *ado* specification will act like a database table, providing all fields that have been obtained from your ADO Recordset.

DAO

1. Click the DAO option in the Select Data Source dialog box. This option allows you to connect to a database file through Data Access Objects (DAO).

2. Select a database type from the Database drop-down list box. This list displays all DAO compatible database drivers installed on your system. Crystal Reports installs many DAO drivers for you. For this example, you can select Access as the database type.

3. Use the **Browse** button to open the Select Database File dialog box. Use this dialog box to locate and select a database file. Crystal Reports includes several sample databases in the \Program Files\Seagate Software\Crystal Reports directory by default. You can select the XTREME.MDB Access file from this directory for this example.

4. Click **Open** in the Select Database File dialog box, and the path and file name of the database you selected appear in the DAO text box on the Select Data Source dialog box.

5. Click **Next**, and the Select Recordset dialog box appears.

6 If the database requires log on information, specify a user name and password to log on.

7 Determine if you want to create a Recordset using an object available from your database, such as a database table, or if you prefer to specify a SQL statement. Select the appropriate option in the Recordset section of the Select Recordset dialog box.

8 If you want to connect to a database object, use the Object Type drop-down list to select the type of database object, such as a Table, then select the object itself from the Object drop-down list.

9 If you want to obtain a Recordset using a SQL statement, write the SQL statement in the text box provided and click Next.

10 Click **Finish** in the Select Recordset dialog box. You will see *dao* in the list box on the Data Tab of the Standard Report Expert.

11 Continue creating your report normally. While creating your report, the *dao* specification will act like a database table, providing all fields that have been obtained from your DAO Recordset.

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

Crystal Reports also supports the "Crystal Data Source Type Library" on page 579, 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 Active Data 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 Driver" on page 565, 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 DAO 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 577
- "Add Fields to the Rowset Object" on page 577
- "Obtain Data as Rows" on page 577
- "Add Rows to the Rowset Object" on page 578

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 569.

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 589.

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 568.

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 589.

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

The Crystal Data Source interface can be implemented inside almost any type of application. You might want to create and internal data source, for instance, inside the same standard executable application that you are implementing the Report Designer Component or another of the Crystal Report development tools. See the "Visual Basic Solutions" chapter in the *Crystal Reports Technical Reference Guide*, or the *Crystal Reports Developer's Help (CrystalDevHelp.chm)* for more information on the other Crystal Reports development tools.. 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

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 Active Data 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 we can use 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.

- From the Project menu, select **Add Class Module**. The Add Class Module dialog box appears.
- 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.

- From the Project menu in Visual Basic, select **Add Module**. The New Module dialog box appears.
- Leave the default *Module* type selected, and click Open. A new module, Module1, is added to your project.
- 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 582
- “Viewing in the Object Browser” on page 582
- “Using Implements in the class declaration” on page 583

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 C:\Program Files\Seagate Software\Crystal Reports.
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. Place a check mark in the check box next to *CRDataSource 1.0 Type Library* if one does not appear already.
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

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.
4. Notice that each of the procedures has been defined as *Private*. For our ActiveX DLL to expose these properties and methods to other applications, we need to change these to *Public*. 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 CRDDataSource 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 sample data DataSource. Obviously, this example has little value in a real application; an ADO Recordset can itself be reported on through the Active Data Driver. However, the example does illustrate how the properties and methods in the Crystal Data Source interface work.

```
Implements CRDataSourceLib.CRDataSource
Dim adoRs As ADOR.Recordset
Private Sub Class_Initialize()
    Set adoRs = New ADOR.Recordset
    adoRs.Open "Customer", "Xtreme sample data", _
        adOpenKeyset, adLockOptimistic, adCmdTable
End 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 that 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, and 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, we will use the ActiveX DLL we created to obtain a MyDataSource object that we can pass to the Active Data Driver in a report generated using the Crystal Designer Component.

For this example, we will assume you have created an application in Visual Basic and designed a report using the Crystal Report Designer Component. For more information on the Crystal Report Designer Component, see the *Crystal Reports Developer's Guide*.

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 Name*, and *City* 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

With your application open in Visual Basic:

1. Choose **References** from the Project menu. The References dialog box appears.
2. Scroll through the list of Available References to locate the *MyDataSourcePrj* component.
3. Add a check mark to 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 Driver" on page 564. 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:

```
Dim myDs As New MyDataSourcePrj.MyDataSource
```

3 In the Form_Load procedure, add the following line before the Report object is assigned to the ReportSource property of the CRViewer1 object:

```
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 65.

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.

Crystal Data Source Object Models 8

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 575. 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” on page 20 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

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

```
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, etc.

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" on page 591.

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, etc.

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” on page 591.

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, etc. 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, etc.

Returns

A Boolean value indicating whether or not the current record was successfully set to the specified record number. This method returns False is 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. One 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. Additionally, a data source defined using the Crystal Data Source Type Library can be passed to report files at runtime through the "Grid Controls and the Crystal Report Engine" on page 20, allowing dynamic runtime reporting on powerful customized data sources.

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 579.

The following topics are discussed in this section:

CRDataSource Properties

The Crystal Data Source interface defines the following properties. An example of each property follows the property description.

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, etc.

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, etc.

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, etc.

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 600
- "MoveNext" on page 600

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" on page 598 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
```

The Crystal Active Data Driver Reference 9

This chapter provides information on the functions available in the Active Data Driver (P2smon.dll). These functions simplify the creation of Data Definition Files and allow you to pass a recordset to a report using the Crystal Report Print Engine.

Overview

The Crystal Active Data Driver provides functions that can be called from your application to simplify the process of designing field definition files and reports at runtime. In addition, a function is provided that translates a Visual Basic object type, such as a Recordset object, into a string value that can be used with the PEGetNthTableLocation function in the Report Engine API.

Declarations of these functions are provided in both C and Visual Basic syntax. If you are using a language other than C, C++, or Visual Basic, refer to the documentation for your development environment for instructions on how to translate C declare statements.

The following topics are discussed in this section:

CreateFieldDefFile

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 Iunknown 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

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

The SetActiveDataSource Function 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” on page 429.

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, for 32-bit environments, in an Automation Server. This section demonstrates how to create User-Defined Functions in a Dynamic Link Library using the C programming language. For information on how to create User-Defined Functions in an Automation Server using Visual Basic or Delphi, see "Creating User-Defined Functions in Visual Basic" on page 623 or "Creating User-Defined Functions in Delphi 3.0" on page 635.

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, and
- 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 606
- "Purpose of the function" on page 607
- "Function data" on page 607
  the report data or user supplied data that the function will require.
- "Return types" on page 608
  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	C Data Type
number	Double.
currency	Double.
Boolean	Short integer.
string	Pointer to an array of characters.
range (number)	Structure containing two doubles.
range (currency)	Structure containing two doubles.
range (Boolean)	Structure containing two short integers.
range (date)	Structure containing two long integers.
range (string)	Structure containing pointers to two elements in a character array.
array (number)	Pointer to a number array.
array (currency)	Pointer to a number array.
array (integer)	Pointer to an integer array.
array (Boolean)	Pointer to a Boolean array.
array (date)	Pointer to a date array.

Parameter Type	C Data Type
array (string)	Pointer to a string array.

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	C Data Type
string	Pointer to a character array.
number	Double.
date	Long integer.
Boolean	Short integer.
currency	Double.
range (date)	Structure containing two long integers.
range (number)	Structure containing two doubles.
range (currency)	Structure containing two doubles.
range (string)	Structure containing pointers to two elements in a character array.
array (date)	Pointer to a date array.
array (number)	Pointer to a number array.
array (currency)	Pointer to a number array.
array (string)	Pointer to a string array.
array (Boolean)	Pointer to a Boolean array.

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 U2L prefix for 32-bit Crystal Reports. For example, U2LSAMP.DLL.
- it must export your User-Defined Function (UDF) as a DLL function, and
- 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” on page 609, 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" on page 619, 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" on page 617, later in the chapter.

Helper Modules

The files listed below have been installed on your system in the same directory as CRW.EXE (C:\CRW by default).

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 620:

File Name	Purpose
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 620, and prototypes of the "InitForJob Function" on page 614, "TermForJob Function" on page 615, and FindJobInfo (see "Implement InitJob and TermJob" on page 622, implemented in UFJOB.C. structure and PROTOTYPES.C.

File Name	Purpose
ufjob.c	Optional file. Contains implementations of the required "InitForJob Function" on page 614 and "TermForJob Function" on page 615. 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" on page 609, into your new directory. This assures 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.

Now, open your Integrated Development Environment (IDE) application, and use the tools there to create a new project file. Name the new project file with a UFL prefix if your function library will be used with Crystal Reports for Windows 3.x, or with a U2L prefix if it will be used with Crystal Reports for Win32. For example, UFLSAMP.MAK or U2LFUNCS.PRJ. Make sure the project is set to build a DLL (not a Windows EXE) file.

Note: If you are building a 32-bit version of 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.

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 619. To begin building your UFL, create a new C module in your IDE, and save it to your working directory with the same name as your project file. For example, if your project file is named UFLSAMP.MAK or UFLFUNCS.PRJ, you would name your C module UFLSAMP.C or UFLFUNCS.C respectively. 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 611,
  - "Function definition table example" on page 612
- "Function Templates Table" on page 612,
- "Function Examples Table" on page 613,
- "Error Table" on page 614,
- "InitForJob Function" on page 614,
- "TermForJob Function" on page 615, and
- "UFL Function Implementation" on page 615.

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), and
- the name of the C/C++ function that implements the call. The definition must appear with the following format:

  "returnType UDFName (arg1, arg2,...)", CFunctionName

Here is an example:

```
"String Picture (String, String)", Picture
```

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.

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:

```
UFFunctionDefStrings FunctionDefStrings [] =
```

Note: The table must be terminated with three nulls.

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 for the Picture function:

```
UFFunctionDefStrings FunctionDefStrings [] =
{
    {“String Picture(String,String)”,Picture},
    {NULL,NULL,NULL}
};
```

An optional third parameter can be used when you know the maximum length of the return value (or your code can obtain it).

For example, the definition for the Picture function might be:

```
{“String Picture (String, String)”, Picture, PictureRetSize},
```

Here, PictureRetSize specifies the maximum acceptable 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, and
- 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:

```
UFFunctionTemplates 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, etc. 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 UFError, defined in ufdll.h. Return UFNoError if no error occurred. Return one of the other UFError 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 ReturnValue.UFReturnUserError member of the parameter block to an error code value you have defined and return UFUserError 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 616, 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 615.

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" on page 614, 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.

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 622, which you can use to retrieve the job information for any open job. "UFJOB Modules" on page 620, 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 610), 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 UFError FAR _export PASCAL FunctionName
    (UFParamBlock *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 UFError. In the error trapping sections of your function, you can return a UFError 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 UFParamBlock 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 UFError 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, however, 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, and
- 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 617), 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:

```
/*******************************
** UFLSAMP.C
**
** Private C code module
** implementing the Picture UDF.
*******************************/

#include <Windows.h>
#include “UFDll.h”
#include “UFMain.h”
#include “UFUser.h”

#define PicturePlaceholder 'x'

/* UDF PROTOTYPE */
ExternC UFError FAR _export 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.
*****/


UFFunctionDefStrings FunctionDefStrings [] =
{
    {“String Picture (String, String)”,
     Picture},
    {NULL, NULL, NULL}
```

```
};

/*****
*  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 UFError FAR _export PASCAL Picture
    (UFParamBlock * ParamBlock)
{
    UFParamListElement*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 UFNoError;
}
```

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 UFMAIN.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 617):

```
Library UFLSAMP
Description'User Function Library for Crystal Reports'
ExeType Windows
HeapSize1024
Code        Moveable Discardable Preload
Data        Moveable Preload
Segments_TEXTPreload

Exports
            WEP
            UFINITIALIZE
            UFTERMINATE
            UFGETVERSION
            UFSTARTJOB
            UFENDJOB
            UFGETFUNCTIONDEFSTRINGS
            UFGETFUNCTIONEXAMPLES
            UFGETFUNCTIONTEMPLATES
            UFERRORRECOVERY
            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 UFMAIN.C. Every UFL must export these 9 UF* functions. In addition to these functions, every UFL that you create must be exported.

Note: If you are creating a 32-bit UFL, do not export the WEP procedure function. If you are creating a 32-bit UFL, you will only need to include the LIBRARY, DESCRIPTION, and EXPORTS sections of the .def file.

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 ufl* or u2l* project file. Resolve any errors that occur, and recompile if necessary. Once you have your DLL (ufl*.dll or u2l*.dll), place it in the directory that holds CRW.EXE. 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 and UFLSAMP2.C (sample files that were installed in the Crystal Reports directory, \CRW, by default).

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” on page 614,and “TermForJob Function” on page 615, 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.

The UFLSAMP2.C file, included with Crystal Reports, demonstrate UFLs that group data according to the Top N values. (Crystal Reports can do this automatically for you, but the functions in UFLSAMP2.C will help you understand how to use the functions and the JobInfo structure in the UFJOB modules.)

If you decide to use the UFJOB modules in your own UFL, the following are required:

- "UFJOB.C" on page 621
  - add UFJOB.C to your UFL project file.
  - Note: See also "UFJOB.H" on page 622
- "UFJOB.H" on page 622
  - #include UFJOB.H in your private C code module that implements your UFL functions.
- "Implement InitJob and TermJob" on page 622
  - 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 614, and "TermForJob Function" on page 615, 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" on page 614, "TermForJob Function" on page 615, 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.

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 606. For information on creating User-Defined Functions in an Automation Server using Delphi 3.0, see "Programming User-Defined Functions in Delphi" on page 636.

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 32-bit version of Crystal Reports Professional Edition includes the User Function Library U2LCOM.DLL. This UFL is installed in your \WINDOWS\CRYSTAL 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 custom 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

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 629, and "Using Arrays" on page 630.

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. Click OK when finished.

Compile the Project as an OLE DLL

1. Save the Module file, the Class Module file, and the P3roject file for your project.
2. Choose Make OLE DLL File from the File menu. 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

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 629, and "Using Arrays" on page 630.

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 system's \WINDOWS\CRYSTAL directory.

Using the User-Defined Functions

From Crystal Reports:

1. Create a new report and add tables to the report, 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. Enter valid arguments for the function. For example:

   ```
   ProjectConversionSquare(5)
   ```
7. Click **Check**. Make sure no errors appear in the function.
8. Click **Accept**, and place the formula in your report. 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 632. 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 UFErrorText string property in your Class Module. This property should be defined Public, using code similar to this:

```
Public UFErrorText As String
```

Any time you trap for an error condition, simply set the UFErrorText 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 UFErrorText.

Note: You should not use the UFErrorText 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 UFErrorText 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 UFInitialize () 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.

UFInitialize

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 606. For information on how to create User Defined Functions in an Automation Server using Visual Basic, see "Programming User-Defined Functions in Visual Basic" on page 624.

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 \WINDOWS\CRYSTAL 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.

Delphi 3.0 allows you to design 32-bit Automation Servers easily. As a Delphi programmer, you can design custom 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

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. Make sure Instancing is set to Multiple Instance, and click **OK**. The Type Library Editor appears.
4. 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 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 639.

Continue creating methods for all User Defined Functions you want to appear in the Crystal Reports Formula Editor.

Register the Type Library

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. 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;
```

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

From Crystal Reports:

1. Create a new report and add tables to the report, 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. Enter valid arguments for the function. For example:

   ```
   ProjectConversionSquare(5)
   ```
7. Click **Check**. Make sure no errors appear in the function.
8. Click **Accept**, and place the formula in your report. 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

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 642.

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 UFErrorText string property in your Class Module. This property should be defined Public, using code similar to this:

```
Property UFErrorText: String;
```

Any time you trap for an error condition, simply set the UFErrorText 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 UFErrorText.

Note: You should not use the UFErrorText 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 UFErrorText 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.

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.

International Office Directory A

This appendix provides contact information by region. For technical support information, see the Technical Support Guide.

International Office Directory

The following section lists the contact information for Sales and Product Support at each Seagate Software office, worldwide.

For more information, visit our web site at http://www.seagatesoftware.com.

North and South American Offices

Canada and USA - North & South American Head Office

Seagate Software
840 Cambie Street
Vancouver, BC V6B 4J2
Canada

Sales and General Inquiries

Telephone: 604 681 3435

Toll-Free: 1 800 877 2340

Fax: 604 681 2934

Latin America

Sales and General Inquiries

Telephone: 604 681 3435

Fax: 604 681 2934

Asia/Pacific Offices

Australia

Seagate Software Pty Ltd.
Level 9, 52 Alfred Street
Milsons Point, NSW 2061
Australia

Sales and General Inquiries

Telephone: +61 2 9955 4088

Fax: +61 2 9955 7682

Hong Kong

Seagate Software (HK)
Suite 2603, 26th Floor
Universal Trade Center
3 Arbuthnot Road, Central
Hong Kong

Sales and General Inquiries

Telephone: +852 2575 2576

Fax: +852 2893 2727

Japan

Seagate Software
3F, Bridgestone Bldg
2-13-12 Hirakawa-cho,
Chiyoda-ku
Tokyo 102-0093

Sales and General Inquiries

Telephone: +81 3 5226 3601

Fax:+81 3 5226 3606

Email: sales_jp@img.seagatesoftware.com

Malaysia

Seagate Software
(a subsidiary of Seagate Technology)
Level 36 Menara Citibank
165 Jalan Ampang
Kuala Lumpur 50450
Malaysia

Sales and General Inquiries

Telephone: 603 2169-6307 or 603-2169-6308

Fax: 603-2169-6168

Email : southasia@seagatesoftware.com

Singapore

Seagate Software
14 Science Park Drive
#03-02 The Maxwell
Singapore Science Park
118226 Singapore

Sales and General Inquiries

Telephone: +65 777 0533

Fax: +65 777 8786

Europe

United Kingdom - EMEA and Northern European Head Office

Seagate Software
The Broadwalk
54 The Broadway
Ealing,
London, W5 5JN
United Kingdom

Seagate Software
St. Matthews Court
4 Civic Drive
Ipswich, Suffolk, 1P1-2QA
United Kingdom

Sales and General Inquiries

Telephone: +44 (0) 208 566 2020

Fax: +44 (0) 208 231 0600

Austria

Seagate Software
Frankfurter-Str. 21-25
D-65760 Eschborn
Germany

Sales and General Inquiries

Telephone: +49 (0) 69 9509 6310

Fax: +49 (0) 69 9509 6314

Belgium

Seagate Software
62 bis, Avenue Andre Morizet
Boulogne-Billancourt Cedex
Paris, F-92643
France

Sales and General Inquiries

Telephone: +33 (0) 155 174 082

Fax: +33 (0) 155 174 086

France - SEMEA. Southern Europe and Middle East Head Office

Seagate Software
62 bis, Avenue Andre Morizet
Boulogne-Billancourt Cedex
Paris, 92643
France

Sales and General Inquiries

Telephone: +33 (0) 155 174 082
or: +33 (0) 1 41 10 16 00

Fax: +33 (0) 155 174 086
or: +33 (0) 1 41 10 16 57

Germany - Central European Head Office

Seagate Software GmbH
Frankfurter-Str. 21-25
D-65760 Eschborn
Germany

Sales and General Inquiries

Telephone: +49 (0) 69 95 09 63 10

Fax: +49 (0) 69 95 09 63 14

Ireland

The Broadwalk
54 The Broadway
Ealing, London
W5 5JN England

Sales and General Inquiries

Telephone: +44 (0) 208 566 2020

Fax: +44 (020) 208 231 0600

Italy

Seagate Software
Via Conservatorio 22
20122 Milano
Italy

Sales and General Inquiries

Telephone: +39 027 729 310

Fax: +39 02 772 940

Netherlands

Seagate Software
Hojel City Centre
GebouwD, 4e etage
Graadt Van Roggenweg 328
Postbox 19127

3501 DC Utrecht
Netherlands

Sales and General Inquiries

Telephone: +44 (0) 208 566 2020

Fax: +44 (0) 208 231 0600

Spain

Seagate Software
Paseo de la Castellana 93-4a
28046, Madrid
Spain

Sales and General Inquiries

Telephone: +34 91 555 5198

Fax: +34 915 559 957

Sweden

Seagate Software
Kanalvagen 10C
Upplands Vasby
194 61 Sweden

Sales and General Inquiries

Telephone: +46 (0) 858 771 171

Fax: + 46 (0) 858 771 172

Switzerland

Seagate Software
World Trade Centre
PO Box 112
Leutschenbachstrasse 95
CH-8050 Zurich
Switzerland

Sales and General Inquiries

Telephone: +41 1 308 3922

Fax: +41 1 308 3500

Africa and Middle East

South Africa

Seagate Software
Sandown Village Office Park
East Court
1 Gwen Lane cnr Maude Street
Sandton 2146
South Africa

Postal Address

Seagate Software
PO Box 78720
Sandton 2146
Republic of South Africa

Sales and General Inquiries

Telephone: +27 11 3059700

Fax: +27 11 3059702

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

Z