Seagate Crystal Reports™ 8
Technical Reference Guide
## CONTENTS

**Chapter 1 - Crystal Report Engine** .............................................. 1
  - Introduction to the Crystal Report Engine .................................. 2
  - Before using the Crystal Report Engine in your application ............ 3
  - Using the Crystal Report Engine .................................................. 4
  - Crystal Report Engine API ............................................................ 6
  - Exporting reports ............................................................................ 32
  - Handling preview window events .................................................... 35
  - Distributing Crystal Report Engine applications ............................. 40
  - Additional sources of information .................................................. 40

**Chapter 2 - Visual Basic Solutions** ............................................. 41
  - Using the Crystal Report Engine API in Visual Basic ...................... 42
  - Crystal ActiveX Controls ............................................................... 46
  - Crystal Report Engine Automation Server ...................................... 49
  - Active Data Driver ......................................................................... 56
  - Crystal Data Object ........................................................................ 65
  - Crystal Data Source Type Library .................................................. 69
  - Grid Controls and the Crystal Report Engine ................................... 77

**Chapter 3 - Report Designer Component Object Model** .................. 81
  - Overview of the Report Designer Object Model .............................. 82
  - Objects and Collections .................................................................. 83
  - Enumerated Types ......................................................................... 228

**Chapter 4 - Seagate Crystal Report Engine** ................................. 255
  - Print Engine Functions ................................................................... 256
  - Print Engine Structures .................................................................. 441
  - Microsoft Windows Structures ......................................................... 523
  - Print Engine Constants .................................................................. 531
  - Obsolete Functions, Structures, and Constants ................................ 553
# Chapter 5 - Using Crystal Report Viewers in Applications

- Application Development with Crystal Report Viewers ..................................... 556
- Seagate Crystal Report Viewer for ActiveX .......................................................... 556
- The Crystal Report Viewer Java Bean ................................................................. 563

# Chapter 6 - Report Viewer Object Model

- Report Viewer/ActiveX Object Model Technical Reference ............................... 566
- The Report Viewer/Java Bean Technical Reference ............................................ 593

# Chapter 7 - Crystal Data Source Object Models

- Crystal Data Source Object Models .................................................................... 600
- Crystal Data Objects ............................................................................................ 600
- Crystal Data Source Type Library ....................................................................... 606

# Chapter 8 - Creating User-Defined Functions in C

- Overview of User-Defined Functions in C ......................................................... 612
- Programming User-Defined Functions in C ....................................................... 612
- Programming the UFL ......................................................................................... 614
- Picture Function - a sample UFL function .......................................................... 623
- Module Definition (.def) File ............................................................................ 626
- UFJOB Modules .................................................................................................. 627

# Chapter 9 - Creating User-Defined Functions in Visual Basic

- Overview of User-Defined Functions in Visual Basic ........................................ 630
- Programming User-Defined Functions in Visual Basic ..................................... 630
- Visual Basic and Seagate Crystal Reports ......................................................... 635
- Sample UFL Automation Server ......................................................................... 639

# Chapter 10 - Creating User-Defined Functions in Delphi 3.0

- Overview of User-Defined Functions in Delphi ................................................ 642
- Programming User-Defined Functions in Delphi .............................................. 642
- Delphi and Seagate Crystal Reports .................................................................... 645
- Sample UFL .......................................................................................................... 649
Appendix A - Product Support ............................................. 651
Product registration ....................................................... 652
Product support .............................................................. 652
Web support ................................................................. 653
Newsletter Service ......................................................... 653
Email support ............................................................... 654
Fax support ................................................................. 654
Technical support ......................................................... 654
Seagate Knowledge Link ............................................... 657
Product return policy .................................................. 657
Product replacement policy ........................................... 657

Appendix B - Seagate Software International Office Directory .... 659
North/South American Head Office ................................. 660
Asia/Pacific Offices ....................................................... 661
Europe/Middle East/Africa Offices ................................. 663
1

Crystal Report Engine

What you will find in this chapter...

Introduction to the Crystal Report Engine, Page 2
...including comments about sample applications, SQL and ODBC, and exporting reports.

Before using the Crystal Report Engine in your application, Page 3

Using the Crystal Report Engine, Page 4
...including creating reports, designing the user interface, adding the Crystal Report Engine to your application, and applications in Delphi.

Crystal Report Engine API, Page 6
...including declarations; using the API; Print-Only and Custom-Print Links; and working with parameter values and ranges, section codes including Visual Basic, variable length strings, API structures, subreports and report formats.

Exporting reports, Page 32
...including comments about PEEportTo function and PEEportOptions structure.

Handling preview window events, Page 35

Distributing Crystal Report Engine applications, Page 40

Additional sources of information, Page 40
INTRODUCTION TO THE CRYSTAL REPORT ENGINE

The following topics are included in this introduction:

  * Sample Applications, Page 2
  * SQL and ODBC, Page 3
  * Exporting Reports, Page 3

Besides acting as a powerful stand-alone report creation application, Seagate Crystal Reports 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 Seagate Crystal Reports. As a licensed user of Seagate 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 Runtime File Requirements online Help.*

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

  * Crystal ActiveX Controls, Page 46 (CRYSTL32.OCX)
  * Crystal Report Engine Automation Server, Page 49 (CPEAUT.DLL or CPEAUT32.DLL)
  * Seagate Crystal Visual Component Library in the Technical Reference Supplemental PDF (UCRPE.DCU or UCRPE32.DCU)
  * The Crystal NewEra Class Library in the Technical Reference Supplemental PDF
  * Crystal Report Engine API, Page 6 (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

Seagate 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.


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 Driver, Page 56.

- 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, Page 4 (The reports that your users access.)
Step 2: Designing the user interface that drives the Crystal Report Engine, Page 4.
Step 3: Adding the Crystal Report Engine to your application, Page 5.

See also:
Using the Crystal Report Engine API in Delphi, Page 6

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 Seagate Crystal Reports User’s Guide and Seagate Crystal Reports online Help, 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 Seagate 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 Driver, Page 56.

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, Page 77, 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, Page 46
- Crystal Report Engine Automation Server, Page 49
- Seagate Crystal Visual Component Library in the Technical Reference Supplemental PDF
- The Crystal NewEra Class Library in the Technical Reference Supplemental PDF
- Crystal Report Engine API, Page 6

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 Using the Crystal Report Engine API in Visual Basic, Page 42. Additional information for Delphi programmers is located in Using the Crystal Report Engine API in Delphi, Page 6. 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 Developer’s online Help.
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:

```delphi
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 Seagate Crystal Report Engine, Page 255, includes Delphi declarations for all Report Engine API functions and records. In addition, the Developer’s online Help 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 Developer’s online Help.

CRYSTAL REPORT ENGINE API

The following topics are discussed in this section:

- Declarations for the Crystal Report Engine API (REAPI), Page 8
- Using the Crystal Report Engine API, Page 8
- The Print-Only Link, Page 9
  - PEPrintReport Arguments, Page 9
  - Example code for a Print-Only Link, Page 11
- The Custom-Print Link, Page 12
  - Coding a Custom-Print Link, Page 12
  - Custom-Print Link Step 1: Open the Crystal Report Engine, Page 13
  - Custom-Print Link Step 2: Open a print job, Page 13
  - Custom-Print Link Step 3: Set the output destination, Page 14
  - Custom-Print Link Step 4: Start the print job, Page 15
  - Custom-Print Link Step 5: Close the print job, Page 15
  - Custom-Print Link Step 6: Close the Crystal Report Engine, Page 15
  - A Sample Custom-Print Link, Page 16
  - Code Evaluation, Page 18
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 Seagate 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 Runtime File Requirements online Help.

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)

Seagate 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 Seagate 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++.
- 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 Developer’s online Help for information on dBASE availability.

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:

1. The Print-Only Link, Page 9
2. The Custom-Print Link, Page 12

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 Seagate 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 Seagate 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:

```pascal
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:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reportFilePath</td>
<td>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.</td>
</tr>
<tr>
<td>toDefaultPrinter</td>
<td>If toDefaultPrinter is set to TRUE (1), the report is sent to a printer. The toWindow argument should be set to FALSE.</td>
</tr>
<tr>
<td>toWindow</td>
<td>If toWindow is set to TRUE (1), the report is sent to a preview window. The toDefaultPrinter argument should be set to FALSE.</td>
</tr>
</tbody>
</table>
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, Seagate 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, Seagate 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, Seagate 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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>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.</td>
</tr>
<tr>
<td>left</td>
<td>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.</td>
</tr>
<tr>
<td>top</td>
<td>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.</td>
</tr>
<tr>
<td>width</td>
<td>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.</td>
</tr>
<tr>
<td>height</td>
<td>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.</td>
</tr>
<tr>
<td>style</td>
<td>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.</td>
</tr>
<tr>
<td>parentWindow</td>
<td>Specifies the window handle for the parent window to be used for this preview window.</td>
</tr>
</tbody>
</table>
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, Seagate 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.

```c
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_USEDEFA U LT, CW_USEDEFA U LT,
```
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 Seagate Crystal Reports Formula Editor, however, must be added when the report is created in Seagate 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, Page 13 (PEOpenEngine).
- Custom-Print Link Step 2: Open a print job, Page 13 (PEOpenPrintJob).
- Custom-Print Link Step 3: Set the output destination, Page 14 (PEOutputToPrinter, PEOutputToWindow, or PEExportTo).
Custom-Print Link Step 4: Start the print job, Page 15 (PEStartPrintJob).

Custom-Print Link Step 5: Close the print job, Page 15 (PEClosePrintJob).


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 Seagate Crystal Report Engine, Page 255 or search for functions by name in Developer’s online Help.

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

```c
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, Page 15). 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

```c
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`, 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, Page 15. 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**

```c
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 PESetPrintJob function.*

The following functions are available to establish a print destination:

- **PEOutputToWindow, Page 368**
  Printing a report to a window requires no other print destination code other than the function itself.

- **PEOutputToPrinter, Page 366**
  Printing a report to a printer requires no other print destination code other than the function itself. However, `PESelectPrinter, Page 375`, can be used to select a printer other than the default printer at runtime. The `PESelectPrinter` function uses the Windows structure `DEVMODE` (page 523). For more information on this structure, refer to the Windows SDK.

- **PEExportTo, Page 283**
  The `PEExportTo` function works with the `PEExportOptions Structure` (page 33) 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 `PESetExportOptions, Page 289` 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, Page 437, 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, Page 274, 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 PEOuputToWindow). 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, Page 272. 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 Seagate 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

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

hJob = PEOpenPrintJob("C:\CRW\ORDER.RPT");
if (!hJob)
{
    ReportError(hJob);
    return;
}

if (ToWindow)
{
    bResult = PECoutputToWindow(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.

```c
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.

```c
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.

```c
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.

```c
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.

```c
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 Developer's online Help 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.

```c
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.

```c
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.

```c
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

```c
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 Seagate Crystal Report Engine, Page 255, or search for the functions by name in Developer’s online Help. The function PEGetHandleString is used to retrieve variable length strings generated by different REAPI functions.

```c
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.

```c
eerrorCode = 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 Developer’s online Help or see Error Codes (page 537).

```c
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.

```c
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.

```c
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 Seagate Crystal Reports handles parameter values and ranges.

Before retrieving a parameter current value or range, Call \textit{PEGetParameterValueInfo}, Page 338, to determine what type of value(s) are stored. \textit{PEParameterValueInfo} (page 479), member hasDiscreteValues will contain one of the following three constants.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_DR_HASRANGE</td>
<td>Only ranges are present.</td>
</tr>
<tr>
<td>PE_DR_HASDISCRETE</td>
<td>Only discrete values are present.</td>
</tr>
<tr>
<td>PE_DR_HASDISCRETE&amp;__RANGE</td>
<td>Both discrete values and ranges are present. See guidelines below.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>\textit{PEXXXParameterCurrentValue(s)}</th>
<th>\textit{PEXXXParameterCurrentRange(s)}</th>
</tr>
</thead>
<tbody>
<tr>
<td>PEGetNParameterCurrentValue, Page 311</td>
<td>PEGetNParameterCurrentRanges, Page 310</td>
</tr>
<tr>
<td>PEGetNthParameterCurrentValue, Page 322</td>
<td>PEGetNthParameterCurrentRange, Page 320</td>
</tr>
<tr>
<td>PEAddParameterCurrentValue, Page 263</td>
<td>PEAddParameterCurrentRange, Page 262</td>
</tr>
</tbody>
</table>

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

- \textit{PEParameterValueInfo}.hasDiscreteValues = PE\_DR\_HASRANGE
  - The parameter field contains only ranges.
  - All values will be treated as ranges.
  - Use the \textit{PEXXXParameterCurrentRange(s)} function calls.

- \textit{PEParameterValueInfo}.hasDiscreteValues = PE\_DR\_HASDISCRETE
  - The parameter field contains only discrete values.
  - All values will be treated as discrete values.
  - Use the \textit{PEXXXParameterCurrentValue(s)} function calls.

- \textit{PEParameterValueInfo}.hasDiscreteValues = PE\_DR\_HASDISCRETE\&_\_RANGE
  - 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

The following topics relating to section codes are presented in this section:

Overview, Page 22
Encoding, Page 22
Decoding, Page 24
Section Map, Page 24
Section Codes in Visual Basic, Page 26

Overview

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, Seagate Crystal Reports uses calculated section codes to identify the sections in each report.

In Seagate 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 Seagate 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 Seagate 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:

<table>
<thead>
<tr>
<th>Section Type Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_SECT_REPORT_HEADER</td>
<td>1</td>
<td>Report Header Section</td>
</tr>
<tr>
<td>PE_SECT_PAGE_HEADER</td>
<td>2</td>
<td>Page Header Section</td>
</tr>
<tr>
<td>PE_SECT_GROUP_HEADER</td>
<td>3</td>
<td>Group Header Section</td>
</tr>
<tr>
<td>PE_SECT_DETAIL</td>
<td>4</td>
<td>Detail Section</td>
</tr>
<tr>
<td>PE_SECT_GROUP_FOOTER</td>
<td>5</td>
<td>Group Footer Section</td>
</tr>
<tr>
<td>PE_SECT_PAGE_FOOTER</td>
<td>7</td>
<td>Page Footer Section</td>
</tr>
<tr>
<td>PE_SECT_REPORT_FOOTER</td>
<td>8</td>
<td>Report Footer Section</td>
</tr>
<tr>
<td>PE_ALLSECTIONS</td>
<td>0</td>
<td>All Report Sections</td>
</tr>
</tbody>
</table>

**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:

```c
    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:

```c
    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:

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

1. **PE_SECTION_TYPE** (sectionCode)
2. **PE_GROUP_N** (sectionCode)
3. **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.

```c
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:

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

**Note:** Earlier versions of Seagate 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 Seagate Crystal Reports can run with applications created with the current version.

### Section Map

The following map shows the pattern of section code assignment:

<table>
<thead>
<tr>
<th>Report Header</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>First Section in Report Header Area</td>
</tr>
<tr>
<td>1025</td>
<td>Second Section in Report Header Area</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>1050</td>
<td>Third Section in Report Header Area</td>
</tr>
<tr>
<td>1075</td>
<td>Fourth Section in Report Header Area</td>
</tr>
<tr>
<td>up to 1975</td>
<td>40th Section in Report Header Area</td>
</tr>
</tbody>
</table>

**Page Header**

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>First Section in Page Header Area</td>
</tr>
<tr>
<td>2025</td>
<td>Second Section in Page Header Area</td>
</tr>
<tr>
<td>2050</td>
<td>Third Section in Page Header Area</td>
</tr>
<tr>
<td>2075</td>
<td>Fourth Section in Page Header Area</td>
</tr>
<tr>
<td>up to 2975</td>
<td>40th Section in Page Header Area</td>
</tr>
</tbody>
</table>

**GH1**

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3000</td>
<td>First Section in First Group Header Area</td>
</tr>
<tr>
<td>3025</td>
<td>Second Section in First Group Header Area</td>
</tr>
<tr>
<td>3050</td>
<td>Third Section in First Group Header Area</td>
</tr>
<tr>
<td>3075</td>
<td>Fourth Section in First Group Header Area</td>
</tr>
</tbody>
</table>

**GH2**

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3001</td>
<td>First Section in Second Group Header Area</td>
</tr>
<tr>
<td>3026</td>
<td>Second Section in Second Group Header Area</td>
</tr>
<tr>
<td>3051</td>
<td>Third Section in Second Group Header Area</td>
</tr>
<tr>
<td>3076</td>
<td>Fourth Section in Second Group Header Area</td>
</tr>
</tbody>
</table>

**Details**

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4000</td>
<td>First Section in Details Area</td>
</tr>
<tr>
<td>4025</td>
<td>Second Section in Details Area</td>
</tr>
<tr>
<td>4050</td>
<td>Third Section in Details Area</td>
</tr>
<tr>
<td>4075</td>
<td>Fourth Section in Details Area</td>
</tr>
</tbody>
</table>

**GF1**

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5000</td>
<td>First Section in First Group Footer Area</td>
</tr>
<tr>
<td>5025</td>
<td>Second Section in First Group Footer Area</td>
</tr>
<tr>
<td>5050</td>
<td>Third Section in First Group Footer Area</td>
</tr>
<tr>
<td>5075</td>
<td>Fourth Section in First Group Footer Area</td>
</tr>
</tbody>
</table>

**GF2**

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5001</td>
<td>First Section in Second Group Footer Area</td>
</tr>
<tr>
<td>5026</td>
<td>Second Section in Second Group Footer Area</td>
</tr>
<tr>
<td>5051</td>
<td>Third Section in Second Group Footer Area</td>
</tr>
<tr>
<td>5076</td>
<td>Fourth Section in Second Group Footer Area</td>
</tr>
</tbody>
</table>
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 Seagate 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

Page Footer

<table>
<thead>
<tr>
<th>Section Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7000</td>
<td>First Section in Page Footer Area</td>
</tr>
<tr>
<td>7025</td>
<td>Second Section in Page Footer Area</td>
</tr>
<tr>
<td>7050</td>
<td>Third Section in Page Footer Area</td>
</tr>
<tr>
<td>7075</td>
<td>Fourth Section in Page Footer Area</td>
</tr>
</tbody>
</table>

Report Footer

<table>
<thead>
<tr>
<th>Section Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8000</td>
<td>First Section in Report Footer Area</td>
</tr>
<tr>
<td>8025</td>
<td>Second Section in Report Footer Area</td>
</tr>
<tr>
<td>8050</td>
<td>Third Section in Report Footer Area</td>
</tr>
<tr>
<td>8075</td>
<td>Fourth Section in Report Footer Area</td>
</tr>
</tbody>
</table>
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, Page 303. 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 PEGetNthSortField, Page 327, 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.

```c
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:
1. a handle to the string,
2. a short integer to hold the length of the string, and
3. 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, Page 328.
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.

```c
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.

```c
/*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.

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

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

`PEGetHandleString`, Page 303, 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.

```c
//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, Page 286
- PEGetErrorText, Page 288
- PEGetFormula, Page 291
- PEGetGroupOptions, Page 301
- PEGetGroupSelectionFormula, Page 302
- PEGetNthFormula, Page 318
- PEGetNthGroupSortField, Page 319
- PEGetNthParameterField, Page 324
- PEGetNthSortField, Page 327
- PEGetReportTitle, Page 342
- PEGetSectionFormatFormula, Page 345
- PEGetSelectedPrinter, Page 347
- PEGetSelectionFormula, Page 349
- PEGetSQLQuery, Page 351
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 Developer’s online Help 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 (page 472), 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, Page 363 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, Page 316, and PEGetNthSubreportInSection, Page 330, 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, Page 352 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 (page 498) 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 Seagate Crystal Reports), use the PEOpenSubreport, Page 365 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, Page 368, you should always avoid making any formatting changes to a print job once you call PEStartPrintJob, Page 437. 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, Page 304, 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

The following topics are discussed in this section:

- **PEExportTo overview, Page 32**
- **PEExportOptions Structure, Page 33**
- **Considerations when using the export functions, Page 34**

Using Seagate 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, Page 283** and **PEGetExportOptions, Page 289**.

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, Page 283** function uses a structure, **PEExportOptions** (page 448), 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, Page 437**, 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, Page 281**, 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 format DLLName.
    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
eamples. 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, Page 289 and PEExportOptions (page 448), you must be using the
version of the Crystal Report Engine (CRPE32.DLL) that came with the Professional Edition of Seagate
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 Seagate Crystal Reports from the
version of Seagate 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 Seagate 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.

```c
#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:
{
    PESStopEventInfo * stopEventInfo = 
    (PESStopEventInfo *) 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 PESStartPrintJob
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.
    PEEWindowOptions 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.printWindowButtonClick = 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

Seagate 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 Runtime File Requirements online Help. 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 Developer’s online Help. 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 Developer’s online Help.

If you are working with the Crystal Report Engine API in Visual Basic, refer to Using the Crystal Report Engine API in Visual Basic, Page 42, for information specific to Visual Basic. Delphi programmers can find information specific to using the Crystal Report Engine API with Delphi under Seagate Crystal Visual Component Library in the Technical Reference Supplemental PDF.
2

Visual Basic Solutions

What you will find in this chapter...

Using the Crystal Report Engine API in Visual Basic, Page 42
...including comments regarding opening and closing the Crystal Report Engine, embedded quotes in VB calls, identifying string issues, passing dates and date ranges, hard coded nulls in User Defined Types and VB Wrapper DLL.

Crystal ActiveX Controls, Page 46
...including comments regarding adding and using ActiveX Controls in your project, and upgrading Crystal Custom Controls.

Crystal Report Engine Automation Server, Page 49
...including comments regarding adding, using and distributing Automation Server in your projects, Object name conflicts, preview window events, viewing the Object Library, and sample applications.

Active Data Driver, Page 56
...including comments regarding using Active Data Driver, Data Definition Files, and using ActiveX Data Sources at design time.

Crystal Data Object, Page 65
...including comments regarding Crystal Data Object, the Object Model, and the Crystal Data Source Type Library

Crystal Data Source Type Library, Page 69
...including adding and implementing the Crystal Data Source Type Library, Crystal Data Source Projects, and passing DataSource objects to the Active Data Driver.

Grid Controls and the Crystal Report Engine, Page 77
...including comments regarding Control Properties, bound and formatted bound reports, and a sample application.
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, Page 42
Embedded Quotes in Visual Basic Calls to the Crystal Report Engine, Page 42
Passing Dates/Date Ranges in Visual Basic using the Crystal Report Engine API Calls, Page 43
Identifying String Issues in Visual Basic Links to the Crystal Report Engine, Page 44
Hard-coded Nulls in Visual Basic User Defined Types, Page 45
Visual Basic Wrapper DLL, Page 45
CRPE32.DEP, Page 45

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**

```vbnet
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**

```vbnet
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 Seagate Crystal Reports record selection formula. This is the syntax you would use if you were entering a selection formula directly into Seagate Crystal Reports.

```vbnet
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.
   ```vbnet
   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.
   ```vbnet
   StartYear$ = 1992
   StartMonth$ = 01
   StartDay$ = 01
   ```

---

*Visual Basic Solutions*
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 Seagate 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 Seagate 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:

```vbnet
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, Page 46, or the Crystal Report Engine Automation Server, Page 49. However, if you prefer to work with the API, Seagate 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 Developer’s online Help. 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. Seagate Crystal Reports includes the Crystal ActiveX Control. Use the ActiveX Control to easily add all of the report processing power of Seagate 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.*

*Note: Seagate Crystal Reports also includes a Visual Basic Custom Control (CRYSTAL.VBX). However, the ActiveX Control is based on more advanced technology and provides more features. You should use the ActiveX Control for development of any new Visual Basic projects. To upgrade an existing project to use the ActiveX Control, see Upgrading from the Crystal Custom Control, Page 48. If, for some reason, you choose to use the VBX in your project rather than the ActiveX Control, the VBX has been fully documented in Developer's online Help.*

The following topics are discussed in this section:

- Adding the ActiveX Control to your Project, Page 46
- Using the ActiveX Controls, Page 47
- Upgrading from the Crystal Custom Control, Page 48

**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 Seagate 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 Seagate 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 Seagate 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:

![ActiveX Control Icon](image)

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 Seagate 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),
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 Technical Reference Supplemental 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, Page 188 property or the PrintReport, Page 285 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 Developer’s online Help. Included are examples of how to set each property at runtime.

Upgrading from the Crystal Custom Control

If you are using the Crystal Custom Control (CRYSTAL.VBX) in a Visual Basic project, you can upgrade your project to use the more powerful Crystal ActiveX control. All previous code and settings will be retained when you upgrade your project.

Normally, Visual Basic versions 4.0 and 5.0 automatically upgrade the control used in your project when you simply open the project in the Visual Basic environment. If Visual Basic does not upgrade your Crystal Custom Control correctly, open the VB.INI file in a text editor, such as Notepad, and verify the following settings exist in the appropriate sections.

```
[VBX Conversions32]
crystal.vbx={00025600-0000-0000-C000-000000000046}
   #5.0#0;c:\windows\system\crystl32.ocx
```

Note: The actual path indicated should correspond to the location of the Crystal ActiveX Control. The path on your system may or may not be the same as the path shown here. In addition, each entry should appear on a single line in your VB.INI file.
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 Seagate 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, Page 49
- Using the Automation Server in Visual Basic, Page 50
- Object Name Conflicts, Page 52
- Viewing the Crystal Report Engine Object Library, Page 53
- Handling Preview Window Events, Page 53
- Distributing the Automation Server with Visual Basic Applications, Page 55
- Sample Applications, Page 55

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 Seagate Crystal Reports, the automation server will have already been registered on your system. If you did not select Development Tools, run the Seagate 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, use the following procedure:

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:

1. **Creating an Application Object**, Page 50
2. **Obtaining a Report Object**, Page 51
3. **Using the Report Object**, Page 51
4. **Releasing Objects**, Page 51
5. **Handling Errors**, Page 52

**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:

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

Alternately, you can use the following code:

```vba
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 Seagate Crystal Reports (.RPT) file and opening it with the OpenReport method of the Application object:

```vbnet
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:

```vbnet
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 Seagate 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:

```vbnet
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 Developer’s online Help.

**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:

```vbnet
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:

```vbnet
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:

```vbnet
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, Page 49), 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 Seagate Crystal Reports Technical Reference. There are several features in the Crystal Report Engine Automation Server that are not available with Seagate 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.

Seagate 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:

```vbnet
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 an 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 Technical Reference Supplemental PDF.

Note: In the previous version of Seagate 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 WithEvents 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 Runtime File Requirements online Help.

## Sample Applications

Seagate 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.

**Visual Basic Solutions** 55
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.

**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 Seagate 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, Page 65*. For information on the Crystal Data Source Type Library, see *Crystal Data Source Type Library, Page 69*.

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 Crystal Report Engine 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.
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, Page 65), and the Crystal Data Source Type Library (see Crystal Data Source Type Library, Page 69).

Once the Crystal Active Data Driver obtains the Recordset from the runtime data source, the Crystal Report Engine 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, Page 61.

Using the Active Data Driver

Designing and generating reports using the Crystal Active Data Driver is a straightforward process, but requires several specific steps:

1. Select the design time data source, Page 58
2. Design the Report, Page 58
3. Obtain a Recordset from the Runtime Data Source, Page 59
4. Open the Report, Page 59
5. Pass the Recordset to the Active Data Driver, Page 60
6. Print the Report, Page 60

The following sections demonstrate this process using the Crystal Active Data Driver with the Crystal Automation Server in Visual Basic 4.0.
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 Seagate Crystal Reports:

1. Click New Report in the Seagate Crystal Reports Welcome dialog box, or click the New button on the Seagate Crystal Reports toolbar. The Report Gallery appears.
3. On the Data Tab, scroll down until you see the Active Data button. If this button does not appear on the Data Tab, you have not correctly installed the Active Data Driver. Run Seagate Crystal Reports Setup again.
4. Click the Active Data button, and the Select Data Source dialog box appears.
5. Click the Data Definition option in the Select Data Source dialog box, and click Browse to locate a data definition file. The Select Data Definition File dialog box appears.
6. Locate the ORDERS.TTX file in the \Program Files\Seagate Software\Crystal Reports directory, and click Open. The specified file appears in the text box for the Data Definition option of the Select Data Source dialog box.
7. Click Finish, and orders appears on the Data Tab of the Report Expert in Seagate Crystal Reports.

Note: For information on specifying an OLE DB provider or other ActiveX data source at design time instead of a data definition file, see Using ActiveX Data Sources at Design Time, Page 63.

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 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 Report Expert.
3. Continue designing the report using the 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 Crystal Report Engine.

Either the Crystal Report Engine API or the Crystal Report Engine Automation Server can be used with the Active Data Driver. However, the following tutorials use the Crystal Report Engine Automation Server in Visual Basic 5.0. This section assumes a familiarity with the Crystal Report Engine Automation Server. If you need more information on how to use the automation server, see the Crystal Report Engine Automation Server, Page 49.

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, Page 65. For information on the Crystal Data Source Type Library, see Crystal Data Source Type Library, Page 69.

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, Page 65).

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.

1. 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:
   
   ```vba
   Dim db As New DAO.Database
   Dim rs As DAO.Recordset
   ```

2. Obtain a Database object from the Xtreme database.
   
   ```vba
   Set db = DBEngine.Workspaces(0).OpenDatabase( _
   "c:\Program Files\Seagate Software\Crystal Reports\xtreme.mdb")
   ```

3. Obtain a Recordset object from the Orders table of the Xtreme database.
   
   ```vba
   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 Crystal Report Engine Automation Server to open a report file.

Note: You must add the Crystal Report Engine 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, Page 49.
1. Declare variables for the Application and Report objects that you will obtain from the Crystal Report Engine Object Library in the automation server. This can be handled in the declarations section of a form or module.

```vba
Dim app As CRPEAuto.Application
Dim report As CRPEAuto.Report
```


```vba
Set app = CreateObject("Crystal.CRPE.Application")
```

3. Obtain a Report object by opening the report file you created earlier. This example uses the file ORDERS.RPT.

```vba
Set report = app.OpenReport("c:\reports\Orders.rpt")
```

**Pass the Recordset to the Active Data Driver**

The Recordset object gets passed to the Active Data Driver through the SetPrivateData method of the DatabaseTable object in the Crystal Report Engine Object Library. You must first obtain a DatabaseTable object from the Report object, then you must use the SetPrivateData method to set the report to point at the recordset object for your Active data source. The Crystal Report Engine 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 DatabaseTable object from the Report object:

```vba
Dim reportDb As CRPEAuto.Database
Dim reportTables As CRPEAuto.DatabaseTables
Dim reportTable As CRPEAuto.DatabaseTable
Set reportDb = report.Database
Set reportTables = reportDb.Tables
Set reportTable = reportTables.Item(1)
```

The Item property in the DatabaseTables collection lets you specify which table in the database you are replacing. Since the data definition file acts as a database with a single table, you should pass 1 to the Item property.

Once you have a DatabaseTable object for the Report object, you can pass the Active data source to the Report object using the SetPrivateData method. This method requires two parameters. The first 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 second parameter is the data source itself. For example:

```vba
reportTable.SetPrivateData(3, rs)
```

**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:

```vba
report.PrintOut
```

Once the data source has been set in the report object, runtime reporting can proceed normally. All features of the Crystal Report Engine are available to you. For more information, refer to the sections of this manual appropriate to the Report Engine development tool you are using.
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 Seagate 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, Page 61. Seagate 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 Seagate Crystal Reports.

The following is an example of how fields are defined in a data definition file:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Length</th>
<th>Sample Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order ID</td>
<td>Long</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Customer Name</td>
<td>String 50</td>
<td></td>
<td>Sample string value</td>
</tr>
<tr>
<td>Order Date</td>
<td>Date</td>
<td></td>
<td>Jan 5, 2000</td>
</tr>
<tr>
<td>Order Amount</td>
<td>Currency</td>
<td></td>
<td>$1.00</td>
</tr>
</tbody>
</table>

The Active Data Driver supports the following data types in a data definition file:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLOB</td>
<td>Fields that contain bitmap images.</td>
</tr>
<tr>
<td>Boolean</td>
<td>True/False Boolean value.</td>
</tr>
<tr>
<td>Byte</td>
<td>8-bit integer value.</td>
</tr>
<tr>
<td>Currency</td>
<td>64-bit floating-point value that can include a currency or percent sign.</td>
</tr>
<tr>
<td>Date</td>
<td>Any date/time value. Examples include:</td>
</tr>
<tr>
<td></td>
<td>● Jan 5, 1999</td>
</tr>
<tr>
<td></td>
<td>● 07/11/97 5:06:07</td>
</tr>
<tr>
<td></td>
<td>● 07/11/97</td>
</tr>
<tr>
<td></td>
<td>● 23:30:01</td>
</tr>
<tr>
<td>Long, int32</td>
<td>32-bit integer value.</td>
</tr>
<tr>
<td>Memo</td>
<td>Any string value over 254 characters long. You must indicate the maximum number of characters for the string.</td>
</tr>
<tr>
<td>Number</td>
<td>64-bit floating-point value.</td>
</tr>
<tr>
<td>Short, int16</td>
<td>16-bit integer value.</td>
</tr>
<tr>
<td>String</td>
<td>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.</td>
</tr>
</tbody>
</table>
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, Seagate 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 Data Tab of the Create Report Expert:

1 Scroll down and click the Active Data button. The Select Data Source dialog box appears.
2 Click the Data Definition option in the dialog box to create a report based on a data definition file.
3 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 Seagate 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 Seagate Crystal Reports (or the Crystal Report Engine) 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. Search for Crystal Active Data Driver Reference in Developer’s online Help 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:

   ```vbs
   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, Seagate 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:** For complete information on the functions provided by the Active Data Driver, search for Crystal Active Data Driver Reference in Developer’s online Help.

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 click the Active Data button on the Data Tab of the Report Expert. 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, Page 63
- ADO and OLE DB, Page 64
- DAO, Page 65

**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 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. You must be familiar with ADO to create a proper connection string. The following are examples of an acceptable connection string for ADO:

3 DSN=Xtreme sample data;

4 DATABASE=pubs;DSN=Publishers;UID=sa;Password=;

5 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.

6 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.

7 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.

8 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.
Click Finish in the Select Recordset dialog box. You will see ado in the list box on the Data Tab of the Report Expert.

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

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).

Select a database type from the Database drop-down list box. This list displays all DAO compatible database drivers installed on your system. Seagate Crystal Reports installs many DAO drivers for you. For this example, you can select Access as the database type.

Use the Browse button to open the Select Database File dialog box. Use this dialog box to locate and select a database file. Seagate 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.

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.

Click Next, and the Select Recordset dialog box appears.

If the database requires log on information, specify a user name and password to log on.

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.

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.

If you want to obtain a Recordset using a SQL statement, write the SQL statement in the text box provided and click Next.

Click Finish in the Select Recordset dialog box. You will see dao in the list box on the Data Tab of the Report Expert.

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 Crystal Report Engine. 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 Crystal Report Engine, analysis is instant and up-to-date.

The following topics are discussed in this section:
- CDO vs. the Crystal Data Source Type Library, Page 66
- Using the Crystal Data Object, Page 66
- Crystal Data Object Model, Page 68

CDO vs. the Crystal Data Source Type Library

Seagate Crystal Reports also supports the Crystal Data Source Type Library, Page 69, 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, Page 57, 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:

1. Obtain a CDO Rowset Object, Page 67
2. Add Fields to the Rowset Object, Page 67
3. Obtain Data as Rows, Page 67
4. Add Rows to the Rowset Object, Page 68

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:

```vbnet
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, Page 61.

Fields are added to a CDO Rowset using the AddField method:

```vbnet
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 Object Reference in Developer's online Help.

**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:

```vbnet
Dim Rows(12, 4) 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.

\textit{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 rows or records 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:

\begin{verbatim}
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
\end{verbatim}

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.

\textbf{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:

\begin{verbatim}
CDOSet.AddRows Rows
\end{verbatim}

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 SetPrivateData method in the Crystal Report Engine Automation Server. For complete instructions on doing this, see \textit{Pass the Recordset to the Active Data Driver, Page 60}.

\textbf{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 \textit{Crystal Data Objects} (page 600).
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, Page 69
Adding the type library, Page 71
Implementing the functions, Page 73
Passing the CRDataSource object to the Active Data Driver, Page 75
Crystal Data Source Projects, Page 76

Creating a new project and class

The Crystal Data Source interface can be implemented inside almost any type of application. You might want to create an internal data source, for instance, inside the same standard executable application that you are implementing the Crystal Designer Component or the Automation Server. 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, Page 70
Creating a new project, Page 70
Adding a class module to a project, Page 70
Adding a Sub Main() procedure, Page 71
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 Crystal Designer Component or the Crystal Report Engine to produce reports based on this new ActiveX data source.

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.

1. From the Project menu, select ADD CLASS MODULE. The Add Class Module dialog box appears.
2. Make sure Class Module is selected, and click Open. The new class module is added to your project, and the code window for the module appears.
Adding a Sub Main() procedure

Although a Sub Main() procedure is not required by ActiveX DLLs created in Visual Basic 5.0 or later, you may want to create a Sub Main() procedure to handle initialization processes. Developers working in Visual Basic 4.0 are required to add the Main subroutine to an Automation Server DLL project and specify that the project use Sub Main() as the entry point. If you are creating an ActiveX EXE in Visual Basic 4.0 or later, you should add the Sub Main() procedure to allow your code to determine if it is being started as a stand-alone application or as an out-of-process automation server.

The following steps demonstrate how to add a Sub Main() procedure in Visual Basic versions 5.0 and 6.0. If you add this procedure to the MyDataSource project, you can leave the procedure empty.

1. From the Project menu in Visual Basic, select ADD MODULE. The New Module dialog box appears.
2. Leave the default Module type selected, and click Open. A new module, Module1, is added to your project.
3. In the code window for the new module, add the following code:

   ```vbp
   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, Page 71
- Viewing in the Object Browser, Page 72
- Using Implements in the class declaration, Page 72

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 Seagate 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, Page 73
Implementing procedures, Page 73
Compiling the ActiveX DLL, Page 75

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 CRDataSource interface depends upon the purpose and design of your application or component. To give you an idea of how to implement the procedures, though, the following code sample simply uses an ADO Recordset object connected to the Xtreme 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.

```vbnet
Implements CRDataSourceLib.CRDataSource
Dim adoRs As ADOR.Recordset
Private Sub Class_Initialize()
    Set adoRs = New ADOR.Recordset
```

Visual Basic Solutions
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 Seagate 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, Page 56*. The following steps simply illustrate how to assign the myDs object created above to the Active Data Driver so that a report will use it as the source of data at runtime.

To actually use the MyDataSourcePrj component, you must create an instance of the MyDataSource object, then assign that object to the Report object displayed by your application. Assuming you created a report in your application using the Crystal Designer Component and accepted default settings for adding the Crystal Report Viewer/ActiveX to your project:

1. Open the code window for the form containing the CrystalReport Viewer/ActiveX.
2. In the General Declarations section for the form, add the following code:
3. `Dim myDs As New MyDataSourcePrj.MyDataSource`
4. In the Form_Load procedure, add the following line before the Report object is assigned to the ReportSource property of the CRViewer1 object:
   ```vbscript
   Report.Database.SetDataSource myDs
   ```

*Note: This example is based on a Visual Basic application created using the Crystal Designer Component, not the Crystal Report Engine Automation Server, Page 49. If you are using the Report Engine Automation Server to assign the new data source to the Active Data Driver, refer to the instructions under Pass the Recordset to the Active Data Driver, Page 60 in the section on the Active Data Driver, Page 56.*

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.

GRID CONTROLS AND THE CRYSTAL REPORT ENGINE

In Seagate 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, Page 77
- Crystal ActiveX Control Properties, Page 77
- Creating a Bound Report using the Crystal ActiveX Control, Page 78
- Creating a Formatted Bound Report, Page 79
- Creating a Formatted Bound Report at Runtime, Page 79
- Sample Application, Page 80

Bound Report Driver and Bound Report Files

When using Seagate 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 Seagate 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.
On the Command Button, add the following code for the Click event:

```vbnet
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 Seagate 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 Seagate 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:

```vbnet
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.
Once the report has been exported, you can open it up in Seagate Crystal Reports.

Perform all formatting changes that you want and save the report.

Return to the Visual Basic application and stop it if it is still running.

On the ActiveX Control:

Set the ReportSource to 0 - Report File.

Set the ReportFileName to the .RPT file that you created.

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 Seagate Crystal Reports since the data does not exist outside of the Visual Basic application.

Note: 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

Seagate 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.
Report Designer Component Object Model

What you will find in this chapter...

Overview of the Report Designer Object Model, Page 82
...including Object naming conflicts.
Objects and Collections, Page 83
...including the Objects and Collections, Properties, Methods and Events listed alphabetically.
Enumerated Types, Page 228
**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 Seagate Crystal Reports. For a list of calls that require additional licensing, please see the Royalty help file (Royalty Required Runtime.hlp).*

- A...D, Page 83
- E...O, Page 85
- P...R, Page 87
- S...Z, Page 90

**A...D**

- Application Object, Page 92
  - Application Object Methods, Page 93
    - `CanClose Method (Application Object)`, Page 93
    - `GetVersion Method (Application Object)`, Page 93
    - `LogOffServer Method (Application Object)`, Page 94
    - `LogOnServer Method (Application Object)`, Page 95
    - `LogOnServerEx Method (Application Object)`, Page 95
    - `NewReport Method (Application Object)`, Page 96
    - `OpenReport Method (Application Object)`, Page 96
    - `SetMatchLogOnInfo Method (Application Object)`, Page 97
    - `SetMorePrintEngineErrorMessage Method (Application Object)`, Page 97
<table>
<thead>
<tr>
<th>DatabaseFieldDefinition Object, Page 115</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>DatabaseFieldDefinition Object Properties, Page 115</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>DatabaseFieldDefinitions Collection, Page 116</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>DatabaseFieldDefinitions Collection Properties, Page 116</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>DatabaseTable Object, Page 116</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>DatabaseTable Object Properties, Page 116</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>DatabaseTable Object Methods, Page 117</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>CheckDifferences Method (DatabaseTable Object), Page 117</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>SetDataSource Method (DatabaseTable Object), Page 118</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>SetLogOnInfo Method (DatabaseTable Object), Page 119</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>SetSessionInfo Method (DatabaseTable Object), Page 119</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>SetTableLocation Method (DatabaseTable Object), Page 120</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>TestConnectivity Method (DatabaseTable Object), Page 120</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>DatabaseTables Collection, Page 120</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>DatabaseTables Collection Properties, Page 121</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>ExportOptions Object, Page 122</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>ExportOptions Object Properties, Page 123</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>ExportOptions Object Methods, Page 125</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>PromptForExportOptions Method (ExportOptions Object), Page 126</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Reset Method (ExportOptions Object), Page 126</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>FieldDefinitions Collection, Page 126</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>FieldDefinitions Collection Properties, Page 126</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>FieldDefinitions Collection Methods, Page 126</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Add Method (FieldDefinitions Collection), Page 127</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Delete Method (FieldDefinitions Collection), Page 127</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>FieldMappingData Object, Page 127</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>FieldMappingData Object Properties, Page 127</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>FieldObject Object, Page 128</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>FieldObject Object Properties, Page 128</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>FieldObject Object Methods, Page 132</td>
</tr>
</tbody>
</table>
— SetLineSpacing Method (FieldObject Object), Page 132
— SetUnboundFieldSource Method (FieldObject Object), Page 133

FormattingInfo Object, Page 133
— FormattingInfo Object Properties, Page 133

FormulaFieldDefinition Object, Page 134
— FormulaFieldDefinition Object Properties, Page 134
— FormulaFieldDefinition Object Methods, Page 135
— Check Method (FormulaFieldDefinition Object), Page 135

FormulaFieldDefinitions Collection, Page 135
— FormulaFieldDefinitions Collection Properties, Page 136
— FormulaFieldDefinitions Collection Methods, Page 136
— Add Method (FormulaFieldDefinitions Collection), Page 136
— Delete Method (FormulaFieldDefinitions Collection), Page 137

GraphObject Object, Page 137
— GraphObject Object Properties, Page 137

GroupNameFieldDefinition Object, Page 143
— GroupNameFieldDefinition Object Properties, Page 143

GroupNameFieldDefinitions Collection, Page 144
— GroupNameFieldDefinitions Collection Properties, Page 144

IFieldDefinition Object, Page 144
— IFieldDefinition Object Properties, Page 144

IReportObject, Page 145
— IReportObject Properties, Page 145

LineObject Object, Page 146
— LineObject Object Properties, Page 146

MapObject Object, Page 147
— MapObject Object Properties, Page 147

ObjectSummaryFieldDefinitions Collection, Page 149
— ObjectSummaryFieldDefinitions Collection Properties, Page 149
— ObjectSummaryFieldDefinitions Collection Methods, Page 149
— Add Method (ObjectSummaryFieldDefinitions Collection), Page 149
— Delete Method (ObjectSummaryFieldDefinitions Collection), Page 150

OlapGridObject Object, Page 150
— OlapGridObject Object Properties, Page 150

OleObject Object, Page 151
— OleObject Object Properties, Page 151
— OleObject Object Methods, Page 153

P...R
Page Object, Page 154
— Page Object Properties, Page 154
— Page Object Methods, Page 154
— RenderEPF Method (Page Object), Page 154
— RenderHTML Method (Page Object), Page 155

PageEngine Object, Page 156
— PageEngine Object Properties, Page 156
— PageEngine Object Methods, Page 156
— CreatePageGenerator Method (PageEngine Object), Page 156
— RenderTotallerETF Method (PageEngine Object), Page 157
— RenderTotallerHTML Method (PageEngine Object), Page 158

PageGenerator Object, Page 159
— PageGenerator Object Properties, Page 159
— PageGenerator Object Methods, Page 160
— CreateSubreportPageGenerator Method (PageGenerator Object), Page 160
— DrillOnGraph Method (PageGenerator Object), Page 161
— DrillOnMap Method (PageGenerator Object), Page 162
— DrillOnSubreport Method (PageGenerator Object), Page 162
— Export Method (PageGenerator Object), Page 163
— FindText Method (PageGenerator Object), Page 163
— GetPageNumberForGroup Method (PageGenerator Object), Page 163
— RenderTotallerETF Method (PageGenerator Object), Page 164
— RenderTotallerHTML Method (PageGenerator Object), Page 165
— GetNextRows Method (Report Object), Page 183
— GetReportVariableValue Method (Report Object), Page 184
— OpenSubreport Method (Report Object), Page 184
— PrinterSetup Method (Report Object), Page 185
— PrintOut Method (Report Object), Page 185
— ReadRecords Method (Report Object), Page 186
— SaveAs Method (Report Object), Page 186
— SelectPrinter Method (Report Object), Page 186
— SetDialogParentWindow Method (Report Object), Page 187
— SetReportVariableValue Method (Report Object), Page 187
— Report Object Events, Page 187
— BeforeFormatPage Event (Report Object), Page 188
— FieldMapping Event (Report Object), Page 188
— NoData Event (Report Object), Page 189

ReportObjects Collection, Page 189
— ReportObjects Collection Properties, Page 189

RunningTotalFieldDefinition Object, Page 190
— RunningTotalFieldDefinition Object Properties, Page 190
— RunningTotalFieldDefinition Object Methods, Page 191
— SetEvaluateConditionField Method (RunningTotalFieldDefinition Object), Page 192
— SetNoEvaluateCondition Method (RunningTotalFieldDefinition Object), Page 192
— SetNoResetCondition Method (RunningTotalFieldDefinition Object), Page 192
— SetResetConditionField Method (RunningTotalFieldDefinition Object), Page 192
— SetSecondarySummarizedField Method (RunningTotalFieldDefinition Object), Page 193
— SetSummarizedField Method (RunningTotalFieldDefinition Object), Page 193

RunningTotalFieldDefinitions Collection, Page 193
— RunningTotalFieldDefinitions Collection Properties, Page 194
— RunningTotalFieldDefinitions Collection Methods, Page 194
— Add Method (RunningTotalFieldDefinitions Collection), Page 194
— Delete Method (RunningTotalFieldDefinitions Collection), Page 195
S...Z

Section Object, Page 195
  — Section Object Properties, Page 195
  — Section Object Methods, Page 196
    — AddBlobFieldObject Method (Section Object), Page 197
    — AddBoxObject Method (Section Object), Page 197
    — AddCrossTabObject Method (Section Object), Page 198
    — AddFieldObject Method (Section Object), Page 199
    — AddGraphObject Method (Section Object), Page 199
    — AddLineObject Method (Section Object), Page 200
    — AddPictureObject Method (Section Object), Page 200
    — AddSpecialVarFieldObject Method (Section Object), Page 201
    — AddSubreportObject Method (Section Object), Page 201
    — AddSummaryFieldObject Method (Section Object), Page 202
    — AddTextObject Method (Section Object), Page 202
    — AddUnboundFieldObject Method (Section Object), Page 203
    — DeleteObject Method (Section Object), Page 204
    — ImportSubreport Method (Section Object), Page 204
  — Section Object Events, Page 204
    — Format Event (Section Object), Page 205
Sections Collection, Page 206
  — Sections Collection Properties, Page 206
  — Sections Collection Methods, Page 207
    — Add Method (Sections Collection), Page 207
    — Delete Method (Sections Collection), Page 207
SortField Object, Page 208
  — SortField Object Properties, Page 208
SortFields Collection, Page 208
  — SortFields Collection Properties, Page 208
  — SortFields Collection Methods, Page 209
    — Add Method (SortFields Collection), Page 209
— Delete Method (SortFields Collection), Page 209
SpecialVarFieldDefinition Object, Page 210
— SpecialVarFieldDefinition Object Properties, Page 210
SQLExpressionFieldDefinition Object, Page 211
— SQLExpressionFieldDefinition Object Properties, Page 211
— SQLExpressionFieldDefinition Object Methods, Page 212
— Check Method (SQLExpressionFieldDefinition Object), Page 212
SQLExpressionFieldDefinitions Collection, Page 212
— SQLExpressionFieldDefinitions Collection Properties, Page 212
— SQLExpressionFieldDefinitions Collection Methods, Page 213
— Add Method (SQLExpressionFieldDefinitions Collection), Page 213
— Delete Method (SQLExpressionFieldDefinitions Collection), Page 213
SubreportLink Object, Page 214
— SubreportLink Object Properties, Page 214
SubreportLinks Collection, Page 214
— SubreportLinks Collection Properties, Page 214
— SubreportLinks Collection Methods, Page 215
— Add Method (SubreportLinks Collection), Page 215
— Delete Method (SubreportLinks Collection), Page 215
SubreportObject Object, Page 216
— SubreportObject Object Properties, Page 216
— SubreportObject Object Methods, Page 217
— OpenSubreport Method (SubreportObject Object), Page 217
SummaryFieldDefinition Object, Page 218
— SummaryFieldDefinition Object Properties, Page 218
— SummaryFieldDefinition Object Methods, Page 219
— SetSecondarySummarizedField Method (SummaryFieldDefinition Object), Page 219
— SetSummarizedField Method (SummaryFieldDefinition Object), Page 220
SummaryFieldDefinitions Collection, Page 220
— SummaryFieldDefinitions Collection Properties, Page 220
— SummaryFieldDefinitions Collection Methods, Page 221
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

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

Or,

```vba
' 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

```vba
' 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,

```vba
' 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), Page 93
GetVersion Method (Application Object), Page 93
LogOffServer Method (Application Object), Page 94
LogOnServer Method (Application Object), Page 95
LogOnServerEx Method (Application Object), Page 95
NewReport Method (Application Object), Page 96
OpenReport Method (Application Object), Page 96
SetMatchLogOnInfo Method (Application Object), Page 97
SetMorePrintEngineErrorMessages Method (Application Object), Page 97

CanClose Method (Application Object)

The CanClose method indicates whether or not the Application Object (page 92) 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 (page 175) 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 the version number of the dll.

Syntax

Function GetVersion () As Integer

Returns

- Returns the version number of the dll.
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) (page 95). This method is only valid if you have purchased Seagate Crystal reports 6.0 or later.

Syntax

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

Parameters

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

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 (page 175), 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) (page 570), in the Seagate 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) (page 94), or until the Application Object (page 92), is destroyed and craxdrt.dll is unloaded. This method is only valid if you have purchased Seagate Crystal reports 6.0 or later.

Syntax

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

Parameters

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

*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), Page 94, or until the Application Object (page 92), is destroyed and craxdrt.dll is unloaded. This method is only valid if you have purchased Seagate Crystal reports 6.0 or later.
Syntax

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

Parameters

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

NewReport Method (Application 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 (page 175), 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pFileName</td>
<td>Specifies a string value indicating the file name and path of the report that you want to open.</td>
</tr>
<tr>
<td>OpenMethod</td>
<td>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.</td>
</tr>
</tbody>
</table>

Returns

- Returns an instance of the **Report Object** (page 175), 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**

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

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bl</td>
<td>Specifies whether the option is selected (TRUE).</td>
</tr>
</tbody>
</table>

**SetMorePrintEngineErrorMessages Method (Application Object)**

Use the **SetMorePrintEngineErrorMessages** method to set the global more print engine error messages option.

**Syntax**

```vba
Sub SetMorePrintEngineErrorMessages ( bl As Boolean )
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bl</td>
<td>Specifies whether the option is selected (TRUE).</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>CopiesToPrint</td>
<td>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.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>DetailHeight</td>
<td>Long. Gets the mailing label report detail area height, in twips.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>DetailWidth</td>
<td>Long. Gets multiple column report detail area width, in twips.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>DiscardOtherGroups</td>
<td>Boolean. Gets or sets the discard other groups option.</td>
<td>Read/Write</td>
<td></td>
</tr>
<tr>
<td>EnableHierarchicalGroupSorting</td>
<td>Boolean. Gets or sets group hierarchically flag.</td>
<td>Read/Write</td>
<td>Fromatting idle.</td>
</tr>
<tr>
<td>GroupCondition</td>
<td>CRGroupCondition (page 239). Gets or sets the group condition.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>GroupConditionField</td>
<td>Object. Gets or sets the group condition field.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>GroupIndent</td>
<td>Long. Gets or sets group indent, in twips.</td>
<td>Read/Write</td>
<td>Fromatting idle.</td>
</tr>
<tr>
<td>GroupNumber</td>
<td>Integer. If the area is a group area, this returns the group number. Otherwise, exception is thrown.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>HideForDrillDown</td>
<td>Boolean. Gets or sets hide for drill down option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>HorizontalGap</td>
<td>Long. Gets the horizontal gaps going across page in a multiple column report.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>InstanceIDField</td>
<td>IFieldDefinition Object (page 144). Gets the instance ID field</td>
<td>Read only</td>
<td>Fromatting idle.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Read/Write</td>
<td>Restriction in event handler</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>KeepGroup Together</td>
<td>Boolean. Gets or sets the keep group together option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>KeepTogether</td>
<td>Boolean. Gets or sets the keep area together option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>Kind</td>
<td>CRAreaKind (page 230). Gets which kind of area (for example, Details, Report Header, Page Footer, etc.).</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Name</td>
<td>String. Gets or sets the area name.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>NewPageAfter</td>
<td>Boolean. Gets or sets the new page after options.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>NewPageBefore</td>
<td>Boolean. Gets or sets the new page before option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>NumberOfTopOrBottomNGroups</td>
<td>Integer. Gets or sets the number of top or bottom groups.</td>
<td>Read/Write</td>
<td>Formatting idle.</td>
</tr>
<tr>
<td>Parent</td>
<td>Report Object (page 175). Gets reference to the parent object.</td>
<td>Read Only</td>
<td>None</td>
</tr>
<tr>
<td>ParentIDField</td>
<td>IFieldDefinition Object (page 144). Gets the parent ID field.</td>
<td>Read only</td>
<td>Formatting idle.</td>
</tr>
<tr>
<td>PrintAtBottomOfPage</td>
<td>Boolean. Gets or sets the print at bottom of page option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>RepeatGroupHeader</td>
<td>Boolean. Gets or sets the repeating group header option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>ResetPageNumberAfter</td>
<td>Boolean. Gets or sets the reset page number after option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>Sections</td>
<td>Sections Collection (page 206). Gets a Collection of all the sections in the area.</td>
<td>Read Only</td>
<td>None</td>
</tr>
<tr>
<td>SortDirection</td>
<td>CRSSortDirection (page 251). Gets or sets the group sort direction.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>Suppress</td>
<td>Boolean. Gets or sets the area visibility.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>TopOrBottomNGroupSortOrder</td>
<td>CRTopOrBottomNGroupSortOrder (page 253). Gets or sets the top or bottom N group sort order.</td>
<td>Read/Write</td>
<td>Formatting idle.</td>
</tr>
<tr>
<td>TopOrBottomNSortField</td>
<td>SummaryFieldDefinition Object (page 218). Gets or sets the top or bottom n sort field.</td>
<td>Read/Write</td>
<td>Formatting idle.</td>
</tr>
</tbody>
</table>
Area Object Methods

The following methods are discussed in this section:

SetInstanceIDField Method (Area Object), Page 100
SetParentIDField (Area Object), Page 100

SetInstanceIDField Method (Area Object)

Use SetInstanceIDField method to set an instance ID field.

Syntax

Sub SetInstanceIDField ( InstanceIDField )

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InstanceIDField</td>
<td>Specifies the instance ID field that you want to set.</td>
</tr>
</tbody>
</table>

SetParentIDField (Area Object)

Use SetParentIDField method to set the parent ID field.

Syntax

Sub SetParentIDField ( ParentIDField )

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentIDField</td>
<td>Specifies the parent ID field that you want to set.</td>
</tr>
</tbody>
</table>

Areas Collection

The Areas Collection contains the area objects for every area in the report. Access a specific Area Object (page 98), in the collection using the Item property.
Areas Collection Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Long. Gets the number of areas in the collection.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Item (index)</td>
<td>Area Object (page 98). 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.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Parent</td>
<td>Report Object (page 175). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackColor</td>
<td>OLE_COLOR. Gets or sets the object background color.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>BorderColor</td>
<td>OLE_COLOR. Gets or sets the object border color.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>BottomCropping</td>
<td>Long. Gets or sets bottom cropping size, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>BottomLineStyle</td>
<td>CRLineStyle (page 241). Gets or sets the bottom line style.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td><strong>Property</strong></td>
<td><strong>Description</strong></td>
<td><strong>Read/Write</strong></td>
<td><strong>Restriction in event handler</strong></td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------------------------</td>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------</td>
</tr>
<tr>
<td>CloseAtPageBreak</td>
<td>Boolean. Gets or sets the close border on page break.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Field</td>
<td><strong>DatabaseFieldDefinition Object</strong> (page 115). Gets the database field definition object containing information about the BLOB field.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>HasDropShadow</td>
<td>Boolean. Gets or sets the border drop shadow option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Height</td>
<td>Long. Gets or sets the object height, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>KeepTogether</td>
<td>Boolean. Gets or sets the keep object together option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Kind</td>
<td><strong>CRObjectKind</strong> (page 244). Gets CRObjectKind which specifies the kind of object (for example, box, cross-tab, field, etc.).</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Left</td>
<td>Long. Gets or sets the object upper left position, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>LeftCropping</td>
<td>Long. Gets or sets the left cropping size, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>LeftLineStyle</td>
<td><strong>CLineStyle</strong> (page 241). Gets or sets the left line style.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Name</td>
<td>String. Gets or sets the object name.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>Parent</td>
<td><strong>Section Object</strong> (page 195). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>RightCropping</td>
<td>Long. Gets or sets the right cropping size, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>RightLineStyle</td>
<td><strong>CLineStyle</strong> (page 241). Gets or sets the right line style.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Suppress</td>
<td>Boolean. Gets or sets the object visibility.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Top</td>
<td>Long. Gets or sets the object upper top position, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>TopCropping</td>
<td>Long. Gets or sets the top cropping size, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>TopLineStyle</td>
<td><strong>CLineStyle</strong> (page 241). Gets or sets the top line style.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Width</td>
<td>Long. Gets or sets the object width, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>XScaling</td>
<td>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.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>YScaling</td>
<td>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.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>Bottom</td>
<td>Long. Gets or sets the object lower bottom position, in twips.</td>
<td>Read/Write</td>
<td></td>
</tr>
<tr>
<td>BottomRightSection</td>
<td>Section Object (page 195). Gets the bottom right section.</td>
<td>Read only</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>CloseAtPageBreak</td>
<td>Boolean. Gets or sets the close border on page break option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>CornerEllispeHeight</td>
<td>Long. Gets or sets the corner ellipse height, in twips.</td>
<td>Read/Write</td>
<td></td>
</tr>
<tr>
<td>CornerEllispeWidth</td>
<td>Long. Gets or sets the corner ellipse width, in twips.</td>
<td>Read/Write</td>
<td></td>
</tr>
<tr>
<td>ExtendToBottomOfSection</td>
<td>Boolean. Gets or sets the extend to bottom of section option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>FillColor</td>
<td>OLE_COLOR. Gets or sets the fill color.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>HasDropShadow</td>
<td>Boolean. Gets or sets the border drop shadow option..</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
</tbody>
</table>
CrossTabGroup Object

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

CrossTabGroup Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackColor</td>
<td>OLE_COLOR. Gets or sets the crosstab group background color.</td>
<td>Read/Write</td>
<td>Idle</td>
</tr>
<tr>
<td>Condition</td>
<td>CRGroupCondition (page 239). Gets or sets the crosstab group condition.</td>
<td>Read/Write</td>
<td>Idle</td>
</tr>
</tbody>
</table>
CrossTabGroups Collection

The CrossTabGroups Collection contains CrossTabGroup Objects associated with a report.

CrossTabGroups Collection Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Long. Gets the CrossTabGroup count.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Item (index As Long)</td>
<td>CrossTabGroup Object (page 104). Gets an item from the Collection.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Parent</td>
<td>CrossTabObject Object (page 106). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
</tbody>
</table>

CrossTabGroups Collection Methods

The following methods are discussed in this section:

Add Method (CrossTabGroups Collection), Page 106
Delete Method (CrossTabGroups Collection), Page 106
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>Specifies the field that you want to add as a CrossTabGroup to the Collection.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>Specifies the index of the CrossTabGroup Object that you want to delete from the Collection.</td>
</tr>
</tbody>
</table>

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

CrossTabObject Object Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackColor</td>
<td>OLE_COLOR. Gets or sets the object background color.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Read/Write</td>
<td>Restriction in event handler</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------------</td>
<td>--------------------------------------------------------</td>
</tr>
<tr>
<td>BorderColor</td>
<td>OLE_COLOR. Gets or sets the object border color.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>BottomLineStyle</td>
<td>CRLineStyle (page 241). Gets or sets the bottom line style.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>CloseAtPageBreak</td>
<td>Boolean. Gets or sets the close border on page break option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>ColumnGrandTotal Color</td>
<td>OLE_COLOR. Gets or sets the column grand total color.</td>
<td>Read/Write</td>
<td>Idle</td>
</tr>
<tr>
<td>ColumnGroups</td>
<td>CrossTabGroups Collection (page 105). Gets the column groups Collection.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>EnableKeep ColumnsTogether</td>
<td>Boolean. Gets or sets the enable keep columns together option.</td>
<td>Read/Write</td>
<td>Idle</td>
</tr>
<tr>
<td>EnableRepeatRow Labels</td>
<td>Boolean. Gets or sets the enable repeat row labels option.</td>
<td>Read/Write</td>
<td>Idle</td>
</tr>
<tr>
<td>EnableShowCell Margins</td>
<td>Boolean. Gets or sets the enable show cell margins option.</td>
<td>Read/Write</td>
<td>Idle</td>
</tr>
<tr>
<td>EnableShowGrid</td>
<td>Boolean. Gets or sets the enable show grid option.</td>
<td>Read/Write</td>
<td>Idle</td>
</tr>
<tr>
<td>EnableSuppress ColumnGrand Totals</td>
<td>Boolean. Gets or sets the enable suppress column grand totals option.</td>
<td>Read/Write</td>
<td>Idle</td>
</tr>
<tr>
<td>EnableSuppress EmptyColumns</td>
<td>Boolean. Gets or sets the enable suppress empty columns option.</td>
<td>Read/Write</td>
<td>Idle</td>
</tr>
<tr>
<td>EnableSuppress EmptyRows</td>
<td>Boolean. Gets or sets the enable suppress empty rows option.</td>
<td>Read/Write</td>
<td>Idle</td>
</tr>
<tr>
<td>EnableSuppress RowGrandTotals</td>
<td>Boolean. Gets or sets the enable suppress row grand totals option.</td>
<td>Read/Write</td>
<td>Idle</td>
</tr>
<tr>
<td>HasDropShadow</td>
<td>Boolean. Gets or sets the border drop shadow option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Height</td>
<td>Long. Gets the object height, in twips.</td>
<td>Read only</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Read/Write</td>
<td>Restriction in event handler</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>KeepTogether</td>
<td>Boolean. Gets or sets the keep object together option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Kind</td>
<td>CRObj ectKind (page 244). Gets which kind of object (for example, box, cross-tab, field, etc.).</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Left</td>
<td>Long. Gets or sets the upper left position, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>LeftLineStyle</td>
<td>CRL ineStyle (page 241). Gets or sets the left line style.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Name</td>
<td>String. Gets or sets the object name.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>Parent</td>
<td>Section Object (page 195). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>RightLineStyle</td>
<td>CRL ineStyle (page 241). Gets or sets the right line style.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>RowGrandTotalColor</td>
<td>OLE_COLOR. Gets or sets the row grand total color.</td>
<td>Read/Write</td>
<td>Idle</td>
</tr>
<tr>
<td>RowGroups</td>
<td>CrossTabGroups Collection (page 105). Gets the row groups Collection.</td>
<td>Read Only</td>
<td>None</td>
</tr>
<tr>
<td>SummaryFields</td>
<td>ObjectSummaryFieldDefinitions Collection (page 149). Gets the summary fields.</td>
<td>Read Only</td>
<td>None</td>
</tr>
<tr>
<td>Suppress</td>
<td>Boolean. Gets or sets the object visibility option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Top</td>
<td>Long. Gets or sets the object upper top position, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>TopLineStyle</td>
<td>CRL ineStyle (page 241). Gets or sets top line style.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Width</td>
<td>Long. Gets the object width, in twips.</td>
<td>Read Only</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
</tbody>
</table>
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 (page 82).

Database Object Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Links</td>
<td>TableLinks Collection (page 223). Gets database link collection.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Parent</td>
<td>Report Object (page 175). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Tables</td>
<td>DatabaseTables Collection (page 120). 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).</td>
<td>Read only</td>
<td>None</td>
</tr>
</tbody>
</table>

Database Object Methods

The following methods are discussed in this section:

AddADOCommand Method (Database Object), Page 110
AddOLEDBSource Method (Database Object), Page 110
LogOffServer Method (Database Object), Page 110
LogOnServer Method (Database Object), Page 111
LogOnServerEx Method (Application Object), Page 95
SetDataSource Method (Database Object), Page 113
Verify Method (Database Object), Page 114
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pConnection</td>
<td>Specifies the ADO connection that you want to use.</td>
</tr>
<tr>
<td>pCommand</td>
<td>Specifies the ADO command that you want to use.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pConnectionString</td>
<td>Specifies the connection string for the OLE DB provider.</td>
</tr>
<tr>
<td>pTableName</td>
<td>Specifies the OLE DB database table that you want to add to your report.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pDllName</td>
<td>Specifies the name of the DLL for the server or password protected non-SQL table you want to log on to, for example &quot;PDSODBC.DLL&quot;. 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.</td>
</tr>
<tr>
<td>pServerName</td>
<td>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.</td>
</tr>
<tr>
<td>[pDatabaseName]</td>
<td>Specifies the name for the database used to create the report. See Remarks below.</td>
</tr>
<tr>
<td>[pUserID]</td>
<td>Specifies the User ID number necessary to log on to the server. See Remarks below.</td>
</tr>
<tr>
<td>[pPassword]</td>
<td>Specifies the password necessary to log on to the server.</td>
</tr>
</tbody>
</table>

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 (page 175), 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) (page 570), in the Seagate 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

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pDllName</td>
<td>Specifies the name of the DLL for the server or password protected non-SQL table you want to log on to, for example &quot;PDSODBC.DLL&quot;. 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.</td>
</tr>
<tr>
<td>pServerName</td>
<td>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.</td>
</tr>
<tr>
<td>[pDatabaseName]</td>
<td>Specifies the name for the database used to create the report. See Remarks below.</td>
</tr>
<tr>
<td>[pUserID]</td>
<td>Specifies the User ID number necessary to log on to the server. See Remarks below.</td>
</tr>
<tr>
<td>[pPassword]</td>
<td>Specifies the password necessary to log on to the server.</td>
</tr>
<tr>
<td>[pServerType]</td>
<td>Specifies the password necessary to log on to the server.</td>
</tr>
<tr>
<td>[pConnectionString]</td>
<td>Specifies the connection string.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>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.</td>
</tr>
<tr>
<td>[dataTag]</td>
<td>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.</td>
</tr>
<tr>
<td>[tableNumber]</td>
<td>Specifies the index number of the table to be set. Default value = 1.</td>
</tr>
</tbody>
</table>

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,

  ```vbs
  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

```vbs
Sub Verify ()
```

Remarks

Prior to calling Verify, you can use the CheckDifferences Method (DatabaseTable Object) (page 117) 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 (page 252) 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

<table>
<thead>
<tr>
<th>Property</th>
<th>Definition</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>DatabaseFieldName</td>
<td>String. Specifies the name of the field in the database (for example, Product ID).</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Kind</td>
<td><strong>CRFieldKind</strong> (page 235). Gets which kind of field (for example, database, summary, formula, etc.).</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Name</td>
<td>String. Gets the unique Crystal formula form name of the field within the report as [table.FIELD] (for example, {product.PRODUCT ID}).</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>NextValue</td>
<td>Variant. Gets the field next value.</td>
<td>Read only</td>
<td>Can be read only when top-level Report object is formatting active.</td>
</tr>
<tr>
<td>NumberOfBytes</td>
<td>Integer. Gets the number of bytes required to store the field data in memory.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Parent</td>
<td><strong>Report Object</strong> (page 175). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>PreviousValue</td>
<td>Variant. Gets the field previous value.</td>
<td>Read only</td>
<td>Can be read only when top-level Report object is formatting active.</td>
</tr>
<tr>
<td>TableAliasName</td>
<td>String. Gets the alias for the table containing the field.</td>
<td>Read only</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>Value</td>
<td>Variant. Gets the field current value.</td>
<td>Read only</td>
<td>Can be read only when top-level Report object is formatting active.</td>
</tr>
<tr>
<td>ValueType</td>
<td><strong>CRFieldValueType</strong> (page 235). Gets which type of value is found in the field.</td>
<td>Read only</td>
<td>None</td>
</tr>
</tbody>
</table>
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 (page 115) in the collection using the Item property.

DatabaseFieldDefinitions Collection Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Long. Gets the number of DatabaseFieldDefinition Object (page 115), in the collection.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Item ( index As Long )</td>
<td>DatabaseFieldDefinition Object (page 115). 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.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Parent</td>
<td>DatabaseTable Object (page 116). Reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
</tbody>
</table>

Remarks

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

DatabaseTable Object

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

DatabaseTable Object Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectBuffer String</td>
<td>String. Gets the table connect buffer string.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>DatabaseType</td>
<td>CRDatabaseType (page 231). Gets the database table type</td>
<td>Read only</td>
<td></td>
</tr>
</tbody>
</table>
DatabaseTable Object Methods

The following methods are discussed in this section:

- CheckDifferences Method (DatabaseTable Object), Page 117
- SetDataSource Method (DatabaseTable Object), Page 118
- SetLogOnInfo Method (DatabaseTable Object), Page 119
- SetSessionInfo Method (DatabaseTable Object), Page 119
- SetTableLocation Method (DatabaseTable Object), Page 120
- TestConnectivity Method (DatabaseTable Object), Page 120

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] )
```
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**

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>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.</td>
</tr>
<tr>
<td>[dataTag]</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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,

  ```vbscript
  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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pServerName</td>
<td>Specifies the name of the server or ODBC data source where the database is</td>
</tr>
<tr>
<td></td>
<td>located (that is, CRSS).</td>
</tr>
<tr>
<td>[pDatabaseName]</td>
<td>Specifies the name of the database.</td>
</tr>
<tr>
<td>[pUserID]</td>
<td>Specifies a valid user name for logging on to the data source.</td>
</tr>
<tr>
<td>[pPassword]</td>
<td>Specifies a valid password for logging on to the data source.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pSessionUserID</td>
<td>Specifies the Access userID used to log on to an Access session.</td>
</tr>
<tr>
<td>pSessionPassword</td>
<td>Specifies the session password for Access secured session.</td>
</tr>
</tbody>
</table>

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), Page 119.

**SetTableLocation Method (DatabaseTable Object)**
The SetTableLocation method is used to set the DatabaseTable location, sublocation, and connect buffer string.

**Syntax**
```vba
Sub SetTableLocation ( pLocation As String, pSubLocation As String, pConnectBufferSting As String )
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pLocation</td>
<td>Specifies the location of the database table (file path and name.ext).</td>
</tr>
<tr>
<td>pSubLocation</td>
<td>Specifies the sublocation of the database table.</td>
</tr>
<tr>
<td>pConnectBufferSting</td>
<td>Specifies the connection buffer string.</td>
</tr>
</tbody>
</table>

**Remarks**
For example:
```vba
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**
```vba
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

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Long. Gets the number of database objects in the collection.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Item (index As Long)</td>
<td>DatabaseTable Object (page 116). 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.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Parent</td>
<td>Database Object (page 109). Reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
</tbody>
</table>

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), Page 121
Delete Method (DatabaseTables Collection), Page 122

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pLocation</td>
<td>Specifies the location of the database table that you want to add to the report.</td>
</tr>
<tr>
<td>[pSubLocation]</td>
<td>Specifies the sublocation of the database table that you want to add to the report.</td>
</tr>
<tr>
<td>[pConnectInfo]</td>
<td>Specifies the connection string.</td>
</tr>
<tr>
<td>[tableType]</td>
<td>Specifies the type of database table that you want to add to the report.</td>
</tr>
</tbody>
</table>
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:
  
  ```vba
  object.Add "xtreme.mdb", "Customer"
  ```

### Delete Method (DatabaseTables Collection)

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

**Syntax**

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

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>Specifies the 1-based index number in the Collection of the database table that you want to delete.</td>
</tr>
</tbody>
</table>

**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 (page 175).
## ExportOptions Object Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplicationFileName</td>
<td>String. Gets or sets the destination application file name.</td>
<td>Read/Write</td>
<td>None</td>
</tr>
<tr>
<td>CharFieldDelimiter</td>
<td>String. Gets or sets the character used to separate fields in character separated text formats. This character delimits every field in the file.</td>
<td>Read/Write</td>
<td>None</td>
</tr>
<tr>
<td>CharStringDelimiter</td>
<td>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).</td>
<td>Read/Write</td>
<td>None</td>
</tr>
<tr>
<td>DestinationDLLName</td>
<td>String. Gets the name of the DLL used to export the report to a specific destination.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>DestinationType</td>
<td>CRExportDestinationType (page 233). Gets or sets the export destination type.</td>
<td>Read/Write</td>
<td>None</td>
</tr>
<tr>
<td>DiskFileName</td>
<td>String. Gets or sets the file name if the report is exported to a disk.</td>
<td>Read/Write</td>
<td>None</td>
</tr>
<tr>
<td>ExcelAreaGroupNumber</td>
<td>Integer. Gets or sets the base area group number if the area type is group area when exporting to Excel.</td>
<td>Read/Write</td>
<td>None</td>
</tr>
<tr>
<td>ExcelAreaType</td>
<td>CRAreaKind (page 230). Gets or sets the base area type if not using constant column width when exporting to Excel.</td>
<td>Read/Write</td>
<td>None</td>
</tr>
<tr>
<td>ExcelConstantColumnWidth</td>
<td>Double. Gets or sets the column width when exporting to Excel.</td>
<td>Read/Write</td>
<td>None</td>
</tr>
<tr>
<td>ExcelTabHasColumnHeadings</td>
<td>Boolean. Gets or sets exporting to Excel has column headings option.</td>
<td>Read/Write</td>
<td>None</td>
</tr>
<tr>
<td>ExcelUseConstantColumnWidth</td>
<td>Boolean. Gets or sets export to Excel to use constant column width.</td>
<td>Read/Write</td>
<td>None</td>
</tr>
<tr>
<td>ExcelUseTabularFormat</td>
<td>Boolean. Gets or sets exporting to Excel to use tabular format.</td>
<td>Read/Write</td>
<td>None</td>
</tr>
<tr>
<td>ExcelUseWorksheetFunctions</td>
<td>Boolean. Gets or sets export to Excel to use worksheet functions to represent subtotal fields in the report.</td>
<td>Read/Write</td>
<td>None</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Read/Write</td>
<td>Restriction in event handler</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>Exchange DestinationType</td>
<td><strong>CRExchangeDestinationType</strong> (page 233). Gets or sets the Exchange destination type for reports exported to Exchange folders.</td>
<td>Read/Write</td>
<td>None</td>
</tr>
<tr>
<td>ExchangeFolderPath</td>
<td>String. Gets or sets the path of the Exchange folder for reports exported to Exchange (for example, “MyFolders@Inbox”).</td>
<td>Read/Write</td>
<td>None</td>
</tr>
<tr>
<td>ExchangePassword</td>
<td>String. Sets Exchange password.</td>
<td>Write only</td>
<td>None</td>
</tr>
<tr>
<td>ExchangeProfile</td>
<td>String. Gets or sets a user profile for accessing an Exchange folder for reports exported to Exchange.</td>
<td>Read/Write</td>
<td>None</td>
</tr>
<tr>
<td>FormatDLLName</td>
<td>String. Gets the file name of the DLL corresponding to the export format.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>FormatType</td>
<td><strong>CRExportFormatType</strong> (page 234). Gets or sets the format type for the exported report (for example, text, Excel, etc.).</td>
<td>Read/Write</td>
<td>None</td>
</tr>
<tr>
<td>HTMLFileName</td>
<td>String. Gets or sets the HTML file name for reports exported to HTML format.</td>
<td>Read/Write</td>
<td>None</td>
</tr>
<tr>
<td>LotusNotesComments</td>
<td>String. Gets or sets the destination Lotus Notes comments.</td>
<td>Read/Write</td>
<td>None</td>
</tr>
<tr>
<td>LotusNotesDatabaseName</td>
<td>String. Gets or sets the destination Lotus Notes database name.</td>
<td>Read/Write</td>
<td>None</td>
</tr>
<tr>
<td>LotusNotesFormName</td>
<td>String. Gets or sets the destination Lotus Notes form name.</td>
<td>Read/Write</td>
<td>None</td>
</tr>
<tr>
<td>MailBccList</td>
<td>String. Gets or sets a Blind Carbon Copy (BCC) list for reports e-mailed to a VIM e-mail account.</td>
<td>Read/Write</td>
<td>None</td>
</tr>
<tr>
<td>MailCcList</td>
<td>String. Gets or sets a Carbon Copy (CC) list for reports e-mailed.</td>
<td>Read/Write</td>
<td>None</td>
</tr>
<tr>
<td>MailMessage</td>
<td>String. Gets or sets the e-mail message included with e-mailed reports.</td>
<td>Read/Write</td>
<td>None</td>
</tr>
<tr>
<td>MailSubject</td>
<td>String. Gets or sets the e-mail subject heading for reports being e-mailed.</td>
<td>Read/Write</td>
<td>None</td>
</tr>
<tr>
<td>MailToList</td>
<td>String. Gets or sets the To list for reports being e-mailed.</td>
<td>Read/Write</td>
<td>None</td>
</tr>
</tbody>
</table>
## ExportOptions Object Methods

The following methods are discussed in this section:

- **PromptForExportOptions Method (ExportOptions Object), Page 126**
- **Reset Method (ExportOptions Object), Page 126**

### ExportOptions Object Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumberOfLinesPerPage</td>
<td>Integer. Gets or sets the number of lines to appear per page option for exported reports that are paginated (for example, HTML).</td>
<td>Read/Write</td>
<td>None</td>
</tr>
<tr>
<td>ODBCDataSourceName</td>
<td>String. Gets or sets the ODBC data source for reports exported to ODBC.</td>
<td>Read/Write</td>
<td>None</td>
</tr>
<tr>
<td>ODBCDataSourcePassword</td>
<td>String. Sets the ODBC data source password.</td>
<td>Write only</td>
<td>None</td>
</tr>
<tr>
<td>ODBCDataSourceUserID</td>
<td>String. Gets or sets the user name used to access an ODBC data source for reports exported to ODBC.</td>
<td>Read/Write</td>
<td>None</td>
</tr>
<tr>
<td>ODBCExportTableName</td>
<td>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.</td>
<td>Read/Write</td>
<td>None</td>
</tr>
<tr>
<td>Parent</td>
<td><strong>Report Object</strong> (page 175). Reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>UseReportDateFormat</td>
<td>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.</td>
<td>Read/Write</td>
<td>None</td>
</tr>
<tr>
<td>UseReportNumberFormat</td>
<td>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.</td>
<td>Read/Write</td>
<td>None</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Long. Gets the collection count.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Item (index As Long)</td>
<td>IFieldDefinition Object (page 144). Gets the collection item.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Parent</td>
<td>IReportObject. Reference to the parent object (ReportObject).</td>
<td>Read only</td>
<td>None</td>
</tr>
</tbody>
</table>

FieldDefinitions Collection Methods
The following methods are discussed in this section:

Add Method (FieldDefinitions Collection), Page 127
Delete Method (FieldDefinitions Collection), Page 127
Add Method (FieldDefinitions Collection)
Use Add method to add the specified Field to the FieldDefinitions Collection.

Syntax
Sub Add (Field)

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>Specifies the Field that you want to add to the Collection.</td>
</tr>
</tbody>
</table>

Delete Method (FieldDefinitions Collection)
Use Delete method to remove the specified Field from the FieldDefinitions Collection

Syntax
Sub Delete(Field)

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>Specifies the Field that you want to delete from the Collection.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>FieldName</td>
<td>String. Gets or sets the field name.</td>
<td>Read/Write</td>
<td>None</td>
</tr>
<tr>
<td>MappingToFieldIndex</td>
<td>Integer. Gets or sets the index of the mapping to field index in the field list of the new database.</td>
<td>Read/Write</td>
<td>None</td>
</tr>
</tbody>
</table>
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 (page 189), (for example, ReportObjects.Item(Index), where the index can be the 1-based index number of the item or an Object name).

FieldObject Object Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>TableName</td>
<td>String. Gets or sets field’s table name.</td>
<td>Read/Write</td>
<td>None</td>
</tr>
<tr>
<td>ValueType</td>
<td><strong>CRFieldValueType</strong> (page 235). Gets or sets the value type that is in the field.</td>
<td>Read/Write</td>
<td>None</td>
</tr>
</tbody>
</table>

128 Seagate Crystal Reports Technical Reference
<table>
<thead>
<tr>
<th><strong>Property</strong></th>
<th><strong>Description</strong></th>
<th><strong>Read/Write</strong></th>
<th><strong>Restriction in event handler</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>CurrencySymbol</td>
<td>String. Gets or sets the currency symbol.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>CurrencySymbolType</td>
<td>[CRCurrencySymbolType](page 231). Gets or sets the currency symbol type.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>DateCalendarType</td>
<td>[CRDateCalendarType](page 231). Gets or sets the date calendar type.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>DateEraType</td>
<td>[CRDateEraType](page 232). Gets or sets the date era type.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>DateFirstSeparator</td>
<td>String. Gets or sets the date first separator.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>DateOrder</td>
<td>[CRDateOrder](page 232). Gets or sets the date order.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>DatePrefixSeparator</td>
<td>String. Gets or sets the date prefix separator.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>DateSuffixSeparator</td>
<td>String. Gets or sets the date suffix separator.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>DateWindowsDefaultType</td>
<td>[CRDateWindowsDefaultType](page 232). Gets or sets the date windows default type.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>DayType</td>
<td>[CRDayType](page 232). Gets or sets the day type.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>DecimalPlaces</td>
<td>Integer. Gets or sets the number decimal places.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>DecimalSymbol</td>
<td>String. Gets or sets the decimal symbol.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>DisplayReverseSign</td>
<td>Boolean. Gets or sets the reverse sign option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>EnableTightHorizontal</td>
<td>Boolean. Gets or sets the tight horizontal option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Field</td>
<td>Object. Gets the field definition object (for example, Database FieldDefinition Object, Parameter FieldDefinition Object, etc.).</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>FirstLineIndent</td>
<td>Long. Gets or sets the first line indent.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td><strong>Property</strong></td>
<td><strong>Description</strong></td>
<td><strong>Read/Write</strong></td>
<td><strong>Restriction in event handler</strong></td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------------------------------------------------------</td>
<td>----------------</td>
<td>---------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Font</td>
<td>IFontDisp. Gets or sets the standard OLE font.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>HasDropShadow</td>
<td>Boolean. Gets or sets the border drop shadow option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Height</td>
<td>Long. Gets or sets object height, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>HorAlignment</td>
<td>CRAlignment (page 229). Gets or sets the horizontal alignment.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>HourMinuteSeparator</td>
<td>String. Gets or sets the hour minute separator.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>HourType</td>
<td>CRHourType (page 239). Gets or sets the hour type.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>KeepTogether</td>
<td>Boolean. Gets or sets the keep object together option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Kind</td>
<td>CROBJECTKind (page 244). Gets which kind of object (for example, box, cross-tab, field, etc.).</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>LeadingDayPosition</td>
<td>CRLleadingDayPosition (page 240). Gets or sets the leading day position option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>LeadingDaySeparator</td>
<td>String. Gets or sets the leading day separator.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>LeadingDayType</td>
<td>CRLleadingDayType (page 240). Gets or sets the leading day type.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Left</td>
<td>Long. Gets or sets the object upper left position, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>LeftIndent</td>
<td>Long. Gets or sets the left indent, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>LeftLineStyle</td>
<td>CRLlineStyle (page 241). Gets or sets the left line style.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>LineSpacing</td>
<td>Double. Gets the line spacing.</td>
<td>Read Only</td>
<td>None</td>
</tr>
<tr>
<td>LineSpacingType</td>
<td>CRLineSpacingType (page 241). Gets the line spacing type.</td>
<td>Read Only</td>
<td>None</td>
</tr>
<tr>
<td>MaxNumberOfLines</td>
<td>Integer. Gets or sets the maximum number of line for a string memo field.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>MinuteSecondSeparator</td>
<td>String. Gets or sets minute second separator.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Read/Write</td>
<td>Restriction in event handler</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>MinuteType</td>
<td><strong>CRMinuteType</strong> (page 242). Gets or sets the minute type.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>MonthType</td>
<td><strong>CRMonthType</strong> (page 243). Gets or sets month type.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Name</td>
<td>String. Gets or sets the object name.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>NegativeType</td>
<td><strong>CRNegativeType</strong> (page 243). Gets or sets number negative type.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>NextValue</td>
<td>Variant. Gets the field next value.</td>
<td>Read only</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Parent</td>
<td><strong>Section Object</strong> (page 195). Reference to the parent object.</td>
<td>Read only</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>PmString</td>
<td>String. Gets or sets the PM string.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>PreviousValue</td>
<td>Variant. Gets the field previous value.</td>
<td>Read only</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>RightIndent</td>
<td>Long. Gets or sets the right indent, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>RightLineStyle</td>
<td><strong>CLineStyle</strong> (page 241). Gets or sets the right line style.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>RoundingType</td>
<td><strong>CRRoundingType</strong> (page 249). Gets or sets the number rounding type.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>SecondType</td>
<td><strong>CRSecondType</strong> (page 250). Gets or sets the seconds type.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Suppress</td>
<td>Boolean. Gets or sets the seconds type.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>SuppressIfDuplicated</td>
<td>Boolean. Gets or sets the suppress if duplicate option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>SuppressIfZero</td>
<td>Boolean. Gets or sets the suppress if zero option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>TextColor</td>
<td>OLE_COLOR. Gets or sets the object text color.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>TextFormat</td>
<td><strong>CRTTextFormat</strong> (page 253). Gets or sets the text format option for string memo fields.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>TextRotationAngle</td>
<td><strong>CRRotationAngle</strong> (page 249). Gets or sets the text rotation angle.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>ThousandsSeparators</td>
<td>Boolean. Gets or sets the enable thousands separators option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
</tbody>
</table>
FieldObject Object Methods

The following methods are discussed in this section:
SetLineSpacing Method (FieldObject Object), Page 132
SetUnboundFieldSource Method (FieldObject Object), Page 133

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 )
SetUnboundFieldSource Method (FieldObject Object)

Use SetUnboundFieldSource Method to bind a DataSource to an unbound field.

Syntax

Sub SetUnboundFieldSource ( pUnboundFieldSource As String )

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pUnboundFieldSource</td>
<td>BSTR specifies the datasource, Crystal formula format. See Remarks below.</td>
</tr>
</tbody>
</table>

Remarks

For example:

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

FormattingInfo Object

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

FormattingInfo Object Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsEndOfGroup</td>
<td>Boolean. Gets whether the current formatting section is the end of a group.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>IsRepeatedGroupHeader</td>
<td>Boolean. Gets whether the current formatting section is a repeated group header.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>IsStartOfGroup</td>
<td>Boolean. Gets whether the current formatting section is the start of a group.</td>
<td>Read only</td>
<td>None</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>FormulaField Name</td>
<td>String. Gets the formula field name as it appears in the RDC Dataview Panel.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Kind</td>
<td><strong>CRFieldKind</strong> (page 235). Gets what kind of field (for example, database, summary, formula, etc.).</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Name</td>
<td>String. Gets the unique name of the formula field in Crystal formula format as it would be referenced in the report (for example, {@ExampleFormula}).</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>NextValue</td>
<td>Variant. Gets the field next value.</td>
<td>Read only</td>
<td>Can be read only when top-level Report object is formatting active.</td>
</tr>
<tr>
<td>NumberOfBytes</td>
<td>Integer. Gets the number of bytes required to store the field data in memory.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Parent</td>
<td><strong>Report Object</strong> (page 175). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>PreviousValue</td>
<td>Variant. Gets the field previous value.</td>
<td>Read only</td>
<td>Can be read only when top-level Report object is formatting active.</td>
</tr>
<tr>
<td>Text</td>
<td>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.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
</tbody>
</table>
Remarks
Seagate 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 (page
175). The default syntax is Crystal Report syntax (crCrystalSyntaxFormula).

FormulaFieldDefinition Object Methods
The following methods are discussed in this section:
Check Method (FormulaFieldDefinition Object), Page 135

Check Method (FormulaFieldDefinition Object)
The Check method checks formula for errors (syntax errors).

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pBool</td>
<td>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.</td>
</tr>
<tr>
<td>ppErrorString</td>
<td>Specifies the error message string if the formula contains an error.</td>
</tr>
</tbody>
</table>

FormulaFieldDefinitions Collection
The FormulaFieldDefinitions Collection is a collection of named formulas in the report. Access a specific
FormulaFieldDefinition Object (page 134), in the collection using the Item property.
FormulaFieldDefinitions Collection Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Long. Gets the number of formula field definitions in the collection.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Item (index As Long)</td>
<td>FormulaFieldDefinitions Collection (page 135). Gets collection item. See Remarks below.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Parent</td>
<td>Report Object (page 175). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
</tbody>
</table>

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), Page 136
Delete Method (FormulaFieldDefinitions Collection), Page 137

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>formulaName</td>
<td>Specifies the formula field that you want to add to the Collection.</td>
</tr>
<tr>
<td>Text</td>
<td>Specifies the text of the formula field that you want to add.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>Specifies the formula field that you want to delete from the Collection.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoRangeData2Axis</td>
<td>Boolean. Gets or sets the auto range option for data2 axis. See Remarks below.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>AutoRangeDataAxis</td>
<td>Boolean. Gets or sets the auto range option for data axis. See Remarks below.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>BackColor</td>
<td>OLE_COLOR. Gets or sets the object background color.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>BarSize</td>
<td>CRBarSize (page 230). Gets or sets the bar size.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>BorderColor</td>
<td>OLE_COLOR. Gets or sets the object border color.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>BottomLineStyle</td>
<td>CRLineStyle (page 241). Gets or sets the bottom line style.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>CloseAtPageBreak</td>
<td>Boolean. Gets or sets the close border on page break option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Read/Write</td>
<td>Restriction in event handler</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>ConditionFields</td>
<td>FieldDefinitions Collection (page 126). Gets the condition fields Collection.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>CrossTabObject</td>
<td>CrossTabObject Object (page 106). Gets the crosstab object if this is a CrossTab chart.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Data2AxisDivisionMethod</td>
<td>CRDivisionMethod (page 233). Gets or sets the data2 axis division method.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>Data2AxisDivisionNumber</td>
<td>Long. Gets or sets the data2 axis division number.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>Data2AxisGridline</td>
<td>CRGridlineType (page 238). Gets or sets the data2 axis grid line type.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>Data2AxisNumberFormat</td>
<td>CRNumberFormat (page 243). Gets or sets the data2 axis number format.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>DataAxisDivisionMethod</td>
<td>CRDivisionMethod (page 233). Gets or sets the data axis division method.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>DataAxisDivisionNumber</td>
<td>Long. Gets or sets the data axis division number.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>DataAxisGridline</td>
<td>CRGridlineType (page 238). Gets or sets the data axis grid line type.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>DataAxisNumberFormat</td>
<td>CRNumberFormat (page 243). Gets or sets the data axis number format.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>DataLabelFont</td>
<td>IFontDisp. Gets or sets standard OLE font for chart data labels.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>DataPoint</td>
<td>CRGraphDataPoint (page 236). Gets or sets the graph data points on risers.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>DataTitleFont</td>
<td>IFontDisp. Gets or sets standard OLE font for chart data title.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>DataType</td>
<td>CRGraphDataType (page 237). Returns which type of data used in the graph.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Read/Write</td>
<td>Restriction in event handler</td>
</tr>
<tr>
<td>------------------------------</td>
<td>--------------------------------------------------</td>
<td>------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>DataValueNumber Format</td>
<td>CRNumberFormat (page 243). Gets or sets the data value number format.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>EnableAutoScale DataAxis</td>
<td>Boolean. Gets or sets the data axis auto-scale option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>EnableAutoScale Data2Axis</td>
<td>Boolean. Gets or sets the data2 axis auto-scale option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>EnableForEachRecord</td>
<td>Boolean. Gets or sets the enable for each record option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>EnableShowLegend</td>
<td>Boolean. Gets or sets the show legend option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>EnableSummarize Values</td>
<td>Boolean. Gets or sets the enable summarize values option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>FootNote</td>
<td>String. Gets or sets the footnote.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>FootnoteFont</td>
<td>IFontDisp. Gets or sets standard OLE font for chart footnote.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>GraphColor</td>
<td>crgraphcolor. Gets or sets the graph color.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>GraphDirection</td>
<td>CRGraphDirection (page 237). Gets or sets the graph direction.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>GraphType</td>
<td>CRGraphType (page 237). Gets or sets the graph type.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>GroupAxisGridline</td>
<td>CRGridlineType (page 238). Gets or sets the group axis grid line type.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>GroupLabelFont</td>
<td>IFontDisp. Gets or sets standard OLE font for chart group labels.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>GroupsTitle</td>
<td>String. Gets or sets the groups title.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>GroupTitleFont</td>
<td>IFontDisp. Gets or sets standard OLE font for chart group title.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>HasDropShadow</td>
<td>Boolean. Gets or sets the border drop shadow option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Read/Write</td>
<td>Restriction in event handler</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------------------------------------</td>
<td>------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>Height</td>
<td>Long. Gets or sets the object height, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>IsXAxisTitleByDefault</td>
<td>Boolean. Gets or sets X-axis title default flag.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>IsFootnoteByDefault</td>
<td>Boolean. Gets or sets footnote default flag.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>IsGroupsTitleByDefault</td>
<td>Boolean. Gets or sets groups title default flag.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>IsSeriesTitleByDefault</td>
<td>Boolean. Gets or sets series title default flag.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>IsSubtitleByDefault</td>
<td>Boolean. Gets or sets subtitle default flag.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>IsTitleByDefault</td>
<td>Boolean. Gets or sets title default flag.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>IsYAxisTitleByDefault</td>
<td>Boolean. Gets or sets Y-axis title default flag.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>IsZAxisTitleByDefault</td>
<td>Boolean. Gets or sets Z-axis title default flag.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>KeepTogether</td>
<td>Boolean. Gets or sets keep object together option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Kind</td>
<td>CRObjectKind (page 244). Gets which kind of object (for example, box, cross-tab, etc.).</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Left</td>
<td>Long. Gets or sets the object upper left position, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>LeftLineStyle</td>
<td>CRLineStyle (page 241). Gets or sets the left line style.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>LegendFont</td>
<td>IFontDisp. Gets or sets standard OLE font for legend.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>LegendLayout</td>
<td>CRPieLegendLayout (page 247). Gets or sets the legend layout for a pie chart.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>LegendPosition</td>
<td>CRLegendPosition (page 241). Gets or sets the legend position.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>MarkerShape</td>
<td>CRMarkerShape (page 242). Gets or sets the marker shape.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>MarkerSize</td>
<td>CRMarkerSize (page 242). Gets or sets the marker size.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Read/Write</td>
<td>Restriction in event handler</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------</td>
<td>----------------------------------------------------</td>
</tr>
<tr>
<td>MaxData2AxisValue</td>
<td>Double. Gets or sets data2-axis max value. See Remarks below.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>MaxDataAxisValue</td>
<td>Double. Gets or sets data-axis max value. See Remarks below.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>MinData2AxisValue</td>
<td>Double. Gets or sets data2-axis min value. See Remarks below.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>MinDataAxisValue</td>
<td>Double. Gets or sets data-axis min value. See Remarks below.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>Name</td>
<td>String. Gets or sets the object name.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>Parent</td>
<td>Section Object (page 195). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>PieSize</td>
<td>CRPieSize (page 247). Gets or sets the size for pie charts.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>RightLineStyle</td>
<td>CRLineStyle (page 241). Gets or sets the right line style.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>SeriesAxisGridline</td>
<td>CRGridlineType (page 238). Gets or sets the series axis grid line type.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>SeriesLabelFont</td>
<td>IFontDisp. Gets or sets standard OLE font for chart series labels.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>SeriesTitle</td>
<td>String. Gets or sets the series title.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>SeriesTitleFont</td>
<td>IFontDisp. Gets or sets standard OLE font for chart series title.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>SliceDetachment</td>
<td>CRSliceDetachment (page 250). Gets or sets the slice detachment.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>SubTitle</td>
<td>String. Gets or sets subtitle.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>SubTitleFont</td>
<td>IFontDisp. Gets or sets standard OLE font for chart subtitle.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>SummaryFields</td>
<td>SummaryFieldDefinitions Collection (page 220). Gets the summary fields Collection.</td>
<td>Read only</td>
<td>None</td>
</tr>
</tbody>
</table>
### Property | Description | Read/Write | Restriction in event handler
--- | --- | --- | ---
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.
TopLineStyle | CRLineStyle (page 241). Gets or sets the top line style. | Read/Write | Can be written only when formatting idle or active.
ViewingAngle | CRViewingAngle (page 254). Gets or sets the viewing angle. | Read/Write | Can be written only when formatting idle.
Width | Long. Gets or sets the object width, in twips. | Read/Write | Can be written only when formatting idle or active.
XAxisTitle | String. Gets or sets the X-Axis title. | Read/Write | Can be written only when formatting idle.
Y2AxisTitle | String. Gets or sets the Y2-Axis title. | Read/Write | Can be written only when formatting idle.
YAxisTitle | String. Gets or sets the Y-Axis title. | Read/Write | Can be written only when formatting idle.
ZAxisTitle | String. Gets or sets the Z-Axis title. | Read/Write | Can be written only when formatting idle.

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

<table>
<thead>
<tr>
<th>This property must be set to FALSE...</th>
<th>...or this property will be ignored</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoRangeData2Axis</td>
<td>MaxData2AxisValue</td>
</tr>
<tr>
<td>AutoRangeDataAxis</td>
<td>MaxDataAxisValue</td>
</tr>
<tr>
<td>AutoRangeSeriesAxis</td>
<td>MaxSeriesAxisValue</td>
</tr>
<tr>
<td>AutoRangeData2Axis</td>
<td>MinData2AxisValue</td>
</tr>
<tr>
<td>AutoRangeDataAxis</td>
<td>MinDataAxisValue</td>
</tr>
<tr>
<td>AutoRangeSeriesAxis</td>
<td>MinSeriesAxisValue</td>
</tr>
</tbody>
</table>
GroupNameFieldDefinition Object

The GroupNameFieldDefinition Object provides properties and methods for retrieving information and setting options for 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 (page 128) when the specified field is a group name field or from a GroupNameFieldDefinitions Collection (page 144) retrieved from the GroupNameFields property of the Report Object (page 175).

**GroupNameFieldDefinition Object Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupNameFieldName</td>
<td>String. Gets the group name field name.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>GroupNumber</td>
<td>Integer. If the area is a group, this gets the group number. Otherwise, exception is thrown.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Kind</td>
<td>CRFieldKind (page 235). Gets which kind of field (for example, database, summary, formula, etc.).</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Name</td>
<td>String. Gets the field definition unique formula name in Crystal Report formula syntax.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>NextValue</td>
<td>Variant. Gets the field next value.</td>
<td>Read only</td>
<td>Can be read only when top-level Report object is formatting active.</td>
</tr>
<tr>
<td>NumberOfBytes</td>
<td>Integer. Gets the number of bytes required to store the field data in memory.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Parent</td>
<td>Report Object (page 175). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>PreviousValue</td>
<td>Variant. Gets the field previous value.</td>
<td>Read only</td>
<td>Can be read only when top-level Report object is formatting active.</td>
</tr>
<tr>
<td>Value</td>
<td>Variant. Gets the field current value.</td>
<td>Read only</td>
<td>Can be read only when top-level Report object is formatting active.</td>
</tr>
<tr>
<td>ValueType</td>
<td>CRFieldValueType (page 235). Gets which type of value is found in the field.</td>
<td>Read only</td>
<td>None</td>
</tr>
</tbody>
</table>
**GroupNameFieldDefinitions Collection**

The `GroupNameFieldDefinitions` Collection is a collection of named groups in the report. Access a specific `GroupNameFieldDefinition Object` (page 143) in the collection using the Item property.

### GroupNameFieldDefinitions Collection Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Long. Gets the number of group name field definitions in the Collection.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Item (index As Long)</td>
<td><code>GroupNameFieldDefinition Object</code> (page 143). 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.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Parent</td>
<td><code>Report Object</code> (page 175). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
</tbody>
</table>

**Remarks**

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

**IFieldDefinition Object**

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

### IFieldDefinition Object Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kind</td>
<td><code>CRFieldKind</code> (page 235). Gets field definition kind.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Name</td>
<td>String. Gets field definition unique formula name.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>NextValue</td>
<td>Variant. Gets the field next value.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>NumberOfBytes</td>
<td>Integer. Gets field number of bytes.</td>
<td>Read only</td>
<td>None</td>
</tr>
</tbody>
</table>
The IReportObject Object provides generic properties for the various types of report objects.

### IReportObject Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent</td>
<td>Report Object (page 175). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>PreviousValue</td>
<td>Variant. Gets the field previous value.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>UseCount</td>
<td>Long. Gets the field use count.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Value</td>
<td>Variant. Gets the field current value.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>ValueType</td>
<td>CRFieldValueType (page 235). Gets the field value type.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>BackColor</td>
<td>OLE_COLOR. Gets or sets the object background color.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>BorderColor</td>
<td>OLE_COLOR. Gets or sets the object border color.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>BottomLineStyle</td>
<td>CRLineStyle (page 241). Gets or sets bottom line style.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>CloseAtPageBreak</td>
<td>Boolean. Gets or sets the close border on page break option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>EnableTightHorizontal</td>
<td>Boolean. Gets or sets the enable tight horizontal option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>HasDropShadow</td>
<td>Boolean. Gets or sets the border drop shadow option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Height</td>
<td>Long. Gets or sets object height, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>KeepTogether</td>
<td>Boolean. Gets or sets the keep object together option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Kind</td>
<td>CRObj ectKind (page 244). Gets which kind of object (for example, box, cross-tab, field, etc.).</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Left</td>
<td>Long. Gets or sets the object upper left position, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeftLineStyle</td>
<td>CRLineStyle (page 241). Gets or sets the left line style.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Name</td>
<td>String. Gets or sets the object name.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Parent</td>
<td>Section Object (page 195). Reference to the parent object.</td>
<td>Read only</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>RightLineStyle</td>
<td>CRLineStyle (page 241). Gets or sets the right line style.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Suppress</td>
<td>Boolean. Gets or sets the object visibility.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Top</td>
<td>Long. Gets or sets the object upper top position, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>TopLineStyle</td>
<td>CRLineStyle (page 241). Gets or sets the top line style.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Width</td>
<td>Long. Gets or sets the object width, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
</tbody>
</table>

### LineObject Object Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bottom</td>
<td>Long. Gets or sets the line lower bottom position, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>EndSection</td>
<td>Section Object (page 195). Gets the end section.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>ExtendToBottomOFSection</td>
<td>Boolean. Gets or sets the extend to bottom of section option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>Kind</td>
<td>CRObjekKind (page 244). Gets which kind of object (for example, box, cross-tab, field, etc.).</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Left</td>
<td>Long. Gets or sets the object upper left position, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
</tbody>
</table>
MapObject Object Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackColor</td>
<td>OLE_COLOR. Gets or sets the object background color.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>BorderColor</td>
<td>OLE_COLOR. Gets or sets the object border color.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>BottomLineStyle</td>
<td>CLineStyle (page 241). Gets or sets the bottom line style.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>CloseAtPageBreak</td>
<td>Boolean. Gets or sets the close border on page break option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
</tbody>
</table>

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.
### Property Description

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>HasDropShadow</td>
<td>Boolean. Gets or sets the border drop shadow option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Height</td>
<td>Long. Gets or sets the object height, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>KeepTogether</td>
<td>Boolean. Gets or sets the keep object together option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Kind</td>
<td><strong>CRObjecKind</strong> (page 244). Gets which kind of object (for example, box, cross-tab, field, etc.).</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Left</td>
<td>Long. Gets or sets the object upper left position, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>LeftLineStyle</td>
<td><strong>CRLinestyle</strong> (page 241). Gets or sets the left line style.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Name</td>
<td>String. Gets or sets the object name.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>Parent</td>
<td><strong>Section Object</strong> (page 195). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>RightLineStyle</td>
<td><strong>CRLinestyle</strong> (page 241). Gets or sets the right line style.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Suppress</td>
<td>Boolean. Gets or sets the object visibility.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Top</td>
<td>Long. Gets or sets the object upper top position, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>TopLineStyle</td>
<td><strong>CRLinestyle</strong> (page 241). Gets or sets the top line style.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Width</td>
<td>Long. Gets or sets the object width, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
</tbody>
</table>
ObjectSummaryFieldDefinitions Collection

ObjectSummaryFieldDefinitions is a Collection of SummaryFieldDefinition Object (page 218) Objects.

ObjectSummaryFieldDefinitions Collection Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Long. Gets the number of object summary field definitions in the Collection.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Item (index As Long)</td>
<td>SummaryFieldDefinition Object (page 218). 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.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Parent</td>
<td>IReportObject (page 145). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
</tbody>
</table>

ObjectSummaryFieldDefinitions Collection Methods

The following methods are discussed in this section:

Add Method (ObjectSummaryFieldDefinitions Collection), Page 149
Delete Method (ObjectSummaryFieldDefinitions Collection), Page 150

Add Method (ObjectSummaryFieldDefinitions Collection)

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

Syntax

Sub Add (summaryField)
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>summaryField</td>
<td>Specifies the formula field that you want to add to the Collection.</td>
</tr>
</tbody>
</table>

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>Specifies the object summary field that you want to delete from the Collection.</td>
</tr>
</tbody>
</table>

OlapGridObject Object

The OlapGridObject Object represents an Olap Object in a report.

OlapGridObject Object Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackColor</td>
<td>OLE_COLOR. Gets or sets the object background color.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>BorderColor</td>
<td>OLE_COLOR. Gets or sets the object border color.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>BottomLineStyle</td>
<td>CRLineStyle (page 241). Gets or sets the bottom line style.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>CloseAtPageBreak</td>
<td>Boolean. Gets or sets the close border on page break option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>HasDropShadow</td>
<td>Boolean. Gets or sets the border drop shadow option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Height</td>
<td>Long. Gets the object height, in twips.</td>
<td>Read only</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
</tbody>
</table>
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**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackColor</td>
<td>OLE_COLOR. Gets or sets the object background color.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>BorderColor</td>
<td>OLE_COLOR. Gets or sets the object border color.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>BottomCropping</td>
<td>Long. Gets or sets the bottom cropping size, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
</tbody>
</table>

**OleObject Object Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>KeepTogether</td>
<td>Boolean. Gets or sets the keep object together option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Kind</td>
<td>CRObjectKind (page 244). Gets report object kind.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Left</td>
<td>Long. Gets or sets the object upper left position, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>LeftLineStyle</td>
<td>CRLineStyle (page 241). Gets or sets the left line style.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Name</td>
<td>String. Gets or sets the object name.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Parent</td>
<td>Section Object (page 195). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>RightLineStyle</td>
<td>CRLineStyle (page 241). Gets or sets the right line style.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Suppress</td>
<td>Boolean. Gets or sets the object visibility.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Top</td>
<td>Long. Gets or sets the object upper top position, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>TopLineStyle</td>
<td>CRLineStyle (page 241). Gets or sets the top line style.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Width</td>
<td>Long. Gets the object width, in twips.</td>
<td>Read only</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Read/Write</td>
<td>Restriction in event handler</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------</td>
<td>----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>BottomLineStyle</td>
<td><strong>CLineStyle</strong> (page 241). Gets or sets the bottom line style.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>CloseAtPageBreak</td>
<td>Boolean. Gets or sets the close border on page break option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>FormattedPicture</td>
<td>IPictureDisp. Gets or sets the specified picture during formatting.</td>
<td>Read/Write</td>
<td>Can be read or written only when top-level Report object is formatting active.</td>
</tr>
<tr>
<td>HasDropShadow</td>
<td>Boolean. Gets or sets the border drop shadow option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Height</td>
<td>Long. Gets or sets the object height, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>KeepTogether</td>
<td>Boolean. Gets or sets the keep object together option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Kind</td>
<td><strong>CRObjecKind</strong> (page 244). Gets which kind of object (for example, box, cross-tab, field, etc.).</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Left</td>
<td>Long. Gets or sets the object upper left position, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>LeftCropping</td>
<td>Long. Gets or sets the left cropping size, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>LeftLineStyle</td>
<td><strong>CLineStyle</strong> (page 241). Gets or sets the left line style.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Name</td>
<td>String. Gets or sets the object name.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>Parent</td>
<td><strong>Section Object</strong> (page 195). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>RightCropping</td>
<td>Long. Gets or sets the right cropping size, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>RightLineStyle</td>
<td><strong>CLineStyle</strong> (page 241). Gets or sets the right line style.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Suppress</td>
<td>Boolean. Gets or sets the object visibility.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Top</td>
<td>Long. Gets or sets the object upper top position, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
</tbody>
</table>
The following methods are discussed in this section:

SetOleLocation Method (OleObject Object), Page 153

SetOleLocation Method (OleObject Object)

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

**Syntax**

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

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pLocation</td>
<td>Specifies the location of the OLE Object.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsLastPage</td>
<td>Boolean. Gets whether the page generated is the last page of the report.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>IsMissing</td>
<td>Boolean. Gets whether the page misses the total page count.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>TotalPageCount</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PageNumber</td>
<td>Long. Gets the page number.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Parent</td>
<td>PageGenerator Object (page 159). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
</tbody>
</table>

Page Object Methods

The following methods are discussed in this section:
- RenderEPF Method (Page Object), Page 154
- RenderHTML Method (Page Object), Page 155

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

```vbnet
Function RenderEPF ( resultType As CRRenderResultType )
```
**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResultType</td>
<td>CRRenderResultType (page 248). Specifies whether the page will be rendered</td>
</tr>
<tr>
<td></td>
<td>using strings or arrays. See Remarks below.</td>
</tr>
</tbody>
</table>

**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**

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>includeDrillDownLinks</td>
<td>Indicates whether or not the HTML page will include hyperlinks for drilling-</td>
</tr>
<tr>
<td></td>
<td>down on summary data.</td>
</tr>
<tr>
<td>pageStyle</td>
<td>CRHTMLPageStyle (page 240). Specifies the style of the HTML page to be</td>
</tr>
<tr>
<td></td>
<td>rendered.</td>
</tr>
<tr>
<td>toolbarStyle</td>
<td>CRHTMLToolbarStyle (page 240). Bitwise constant specifies the style of the</td>
</tr>
<tr>
<td></td>
<td>toolbar to be used. Constants can be XOR’d.</td>
</tr>
<tr>
<td>baseURL</td>
<td>The URL used to access the report when it is first generated.</td>
</tr>
<tr>
<td>resultType</td>
<td>CRRenderResultType (page 248). Specifies whether the page will be rendered</td>
</tr>
<tr>
<td></td>
<td>using strings or arrays. See Remarks below.</td>
</tr>
</tbody>
</table>

**Returns**

Returns the HTML stream.

**Remarks**

Currently only arrays are supported.
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

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>ImageOptions</td>
<td>CRImageType (page 240). Gets or sets the image type for EPF format.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>Parent</td>
<td>Report Object (page 175). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>PlaceHolderOptions</td>
<td>CRPlaceHolderType (page 247). Gets or sets the EPF place holder options.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>ValueFormat Options</td>
<td>CRValueFormatType (page 254). Gets or sets the EPF value format options.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
</tbody>
</table>

### PageEngine Object Methods

The following methods are discussed in this section:

- **CreatePageGenerator Method (PageEngine Object)**, Page 156
- **RenderTotallerETF Method (PageEngine Object)**, Page 157
- **RenderTotallerHTML Method (PageEngine Object)**, Page 158

**CreatePageGenerator Method (PageEngine Object)**

The CreatePageGenerator method returns a PageGenerator Object (page 159), 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

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

Returns

Returns a PageGenerator Object (page 159).

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rootGroupPath</td>
<td>Specifies an integer array that represents the group path. An empty array</td>
</tr>
<tr>
<td></td>
<td>would represent the entire report, an array (0, 1) would represent a drill</td>
</tr>
<tr>
<td></td>
<td>down on the second group member within the first group member where 0 = first</td>
</tr>
<tr>
<td></td>
<td>group in Group #1 and 1 = second group in Group #2.</td>
</tr>
<tr>
<td>startingChildNumber</td>
<td>The start child number to display.</td>
</tr>
<tr>
<td>pastRootLevels</td>
<td>Past root levels.</td>
</tr>
<tr>
<td>maxNodeCount</td>
<td>The maximum number of nodes for each group level in the Group Tree.</td>
</tr>
<tr>
<td>resultType</td>
<td>CRRenderResultType (page 248). Specifies whether the page will be rendered</td>
</tr>
<tr>
<td></td>
<td>using strings or arrays. See Remarks below.</td>
</tr>
</tbody>
</table>

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

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rootGroupPath</td>
<td>Specifies the base URL string.</td>
</tr>
<tr>
<td>startingChildNumber</td>
<td>The start child number to display.</td>
</tr>
<tr>
<td>pastRootLevels</td>
<td>A value indicating the number of past root levels.</td>
</tr>
<tr>
<td>maxNodeCount</td>
<td>The maximum number of nodes to display for each group level in the Group Tree.</td>
</tr>
<tr>
<td>openGroupPath</td>
<td>An array of groups to be opened in the report.</td>
</tr>
<tr>
<td>includeDrillDownLinks</td>
<td>Indicates whether or not drill down hyperlinks are generated for summary values in the report.</td>
</tr>
<tr>
<td>baseURL</td>
<td>The URL used to access the report when it is first generated.</td>
</tr>
<tr>
<td>resultType</td>
<td>CRRenderResultType (page 248). Specifies whether the page will be rendered using strings or arrays. See Remarks below.</td>
</tr>
</tbody>
</table>

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 (page 154), as they are requested and allows options for manipulating the report as a whole.

PageGenerator Object Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>ContainingGroupName</td>
<td>String. Gets containing group name for out of place subreport view.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>ContainingGroupPath</td>
<td>Variant. Gets containing group path for out of place subreport view.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>ContainingPageNumber</td>
<td>Long. Gets containing page number for out of place subreport view.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>DrillDownLevel</td>
<td>Ignore. This property is currently reserved.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>GroupName</td>
<td>String. Gets the group name for drill down on graph view.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>GroupPath</td>
<td>Variant. Gets the GroupPath parameter set by CreatePageGenerator Method (PageEngine Object), Page 156.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Pages</td>
<td>Pages Collection (page 166). Gets the collection of pages for a specified view.</td>
<td>Read only</td>
<td>None</td>
</tr>
</tbody>
</table>
PageGenerator Object Methods

The following methods are discussed in this section:

CreateSubreportPageGenerator Method (PageGenerator Object), Page 160
DrillOnGraph Method (PageGenerator Object), Page 161
DrillOnMap Method (PageGenerator Object), Page 162
DrillOnSubreport Method (PageGenerator Object), Page 162
Export Method (PageGenerator Object), Page 163
FindText Method (PageGenerator Object), Page 163
GetPageNumberForGroup Method (PageGenerator Object), Page 163
RenderTotallerETF Method (PageGenerator Object), Page 164
RenderTotallerHTML Method (PageGenerator Object), Page 165

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
DrillOnGraph Method (PageGenerator Object)

The DrillOnGraph method creates a new PageGenerator Object (page 159) 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupPath</td>
<td>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.</td>
</tr>
<tr>
<td>[DrillDownLevel]</td>
<td>Reserved. Do not use.</td>
</tr>
</tbody>
</table>

Returns

Returns a PageGenerator Object (page 159).
DrillOnMap Method (PageGenerator Object)
The DrillOnMap method creates a new PageGenerator Object (page 159), 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PageNumber</td>
<td>Specifies the page number.</td>
</tr>
<tr>
<td>xOffset</td>
<td>Specifies the X coordinate on page, in twips, where the drill down occurred.</td>
</tr>
<tr>
<td>yOffset</td>
<td>Specifies the Y coordinate on page, in twips, where the drill down occurred.</td>
</tr>
</tbody>
</table>

Returns

Returns a PageGenerator Object (page 159).

DrillOnSubreport Method (PageGenerator Object)
The DrillOnSubreport method creates a new PageGenerator Object (page 159), 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PageNumber</td>
<td>Specifies the page number.</td>
</tr>
<tr>
<td>xOffset</td>
<td>Specifies the X coordinate on page, in twips, where the drill down occurred.</td>
</tr>
<tr>
<td>yOffset</td>
<td>Specifies the Y coordinate on page, in twips, where the drill down occurred.</td>
</tr>
</tbody>
</table>

Returns

Returns a PageGenerator Object (page 159).
Export Method (PageGenerator Object)
Use Export method to obtain an export stream.

Syntax

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

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resultType</td>
<td>CRRenderResultType (page 248). Specifies the result type.</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>Specifies the text string that you want to search for.</td>
</tr>
<tr>
<td>direction</td>
<td>CRSearchDirection (page 250). Specifies the direction that you want to search.</td>
</tr>
<tr>
<td>pPageNumber</td>
<td>Specifies the page on which you want to start the search.</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupPath</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rootGroupPath</td>
<td>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.</td>
</tr>
<tr>
<td>startingChildNumber</td>
<td>The starting child level to display the report grouping at.</td>
</tr>
<tr>
<td>pastRootLevels</td>
<td>Specifies the number of past root levels.</td>
</tr>
<tr>
<td>maxNodeCount</td>
<td>The maximum number of nodes for each group level in the Group Tree.</td>
</tr>
<tr>
<td>resultType</td>
<td>CRRenderResultType (page 248). Specifies whether the page will be rendered using strings or arrays. See Remarks below.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rootGroupPath</td>
<td>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.</td>
</tr>
<tr>
<td>startingChildNumber</td>
<td>The starting child level to display the report grouping at.</td>
</tr>
<tr>
<td>pastRootLevels</td>
<td>A value indicating the number of past root levels.</td>
</tr>
<tr>
<td>maxNodeCount</td>
<td>The maximum number of nodes to display for each group level in the Group Tree.</td>
</tr>
<tr>
<td>openGroupPath</td>
<td>An array of groups to be opened in the report.</td>
</tr>
<tr>
<td>includeDrillDownLinks</td>
<td>Indicates whether or not drill down hyperlinks are generated for summary values in the report.</td>
</tr>
<tr>
<td>baseURL</td>
<td>The URL used to access the report when it is first generated.</td>
</tr>
<tr>
<td>resultType</td>
<td>CRRenderResultType (page 248). Specifies whether the page will be rendered using strings or arrays. See Remarks below.</td>
</tr>
</tbody>
</table>

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 (page 154). Access a specific Page Object in the collection using the Item property.

Pages Collection Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Long. Gets the number of page objects in the collection.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Item (index As Long)</td>
<td>Page Object (page 154). Gets the collection item based on the 1-based index number.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Parent</td>
<td>PageGenerator Object (page 159). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
</tbody>
</table>

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 (page 128), when the specified field is a parameter field or from the ParameterFieldDefinitions Collection (page 173) Object under Report.
## ParameterFieldDefinition Object Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>DisallowEditing</td>
<td>Boolean. Gets or sets the disallowing editing option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>DiscreteOrRangeKind</td>
<td><strong>CRDiscreteOrRangeKind</strong> (page 233). Gets or sets the parameter value kind (discrete and/or range).</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>EditMask</td>
<td>String. Gets or sets edit mask for string parameter. See Remarks below for additional information.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>EnableExclusiveGroup</td>
<td>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.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>EnableMultipleValues</td>
<td>Boolean. Gets or sets the allow multiple values option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>EnableNullValue</td>
<td>Boolean. Gets or sets the value nullable option for Stored Procedure parameters.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>EnableRangeLimit</td>
<td>Boolean. Gets or sets the option which specifies if this parameter field value should be in the specified range.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>EnableShowDescriptionOnly</td>
<td>Boolean. Gets or sets the show description for pick list option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>EnableSortBasedOnDesc</td>
<td>Boolean. Gets or sets the sort based on description in pick list option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>GroupNumber</td>
<td>Integer. Gets or sets boolean group number.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>IsCurrentValueSet</td>
<td>Boolean. Gets whether the current value has been set.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>IsDefaultValueSet</td>
<td>Boolean. Gets whether the default value has been set.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Kind</td>
<td>Description</td>
<td>Access</td>
<td>Remarks</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CRFieldKind</td>
<td>(page 235). Gets which kind of field (database, summary, formula, etc.).</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>MaximumValue</td>
<td>Variant. Gets or sets the maximum value. See Remarks below for additional comments.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>MinimumValue</td>
<td>Variant. Gets or sets minimum value. See Remarks below for additional comments.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>Name</td>
<td>String. Gets the unique formula name of the parameter field as it appears in the Parameter Field list (RDC DataView).</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>NeedsCurrentValue</td>
<td>Boolean. Gets whether the field needs a current value.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>NextValue</td>
<td>Variant. Gets the next field value.</td>
<td>Read only</td>
<td>Can be read only when top-level Report object is formatting active.</td>
</tr>
<tr>
<td>NthValue Description (index As Integer)</td>
<td>String. Gets or sets nth value description.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>NumberOfBytes</td>
<td>Integer. Gets the number of bytes required to store the field data in memory.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>NumberOfCurrentRanges</td>
<td>Integer. Gets total number of current ranges.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>NumberOfCurrentValues</td>
<td>Integer. Gets the total number of current values.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>NumberOfDefaultValues</td>
<td>Integer. Gets the total number of default values.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>ParameterFieldName</td>
<td>String. Gets the name of the parameter field as it is displayed (referenced) in the report (RDC DataView).</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>ParameterType</td>
<td>CRParameterFieldType (page 246). Gets parameter type.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Parent</td>
<td>Report Object (page 175). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>PickListSortMethod</td>
<td>CRParameterPickListSortMethod (page 246). Gets or sets the sort method in pick list option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
</tbody>
</table>
### PlaceInGroup

| **Boolean. Gets or sets, when prompting for values, whether a Boolean parameter field should appear as part of a Boolean parameter group or individually. Used in conjunction with Boolean property, EnableExclusiveGroup.** | **Read/Write** | **Can be written only when formatting idle.** |

### PreviousValue

| **Variant. Gets the field previous value.** | **Read only** | **Can be read only when top-level Report object is formatting active.** |

### Prompt

| **String. Gets or sets the parameter field prompting string.** | **Read/Write** | **None** |

### ReportName

| **String. Gets the report name the parameter field is in. If it is a main report, the ReportName is empty.** | **Read only** | **None** |

### Value

| **Variant. Gets the current value of the field.** | **Read only** | **Can be read only when top-level Report object is formatting active.** |

### ValueType

| **CRFieldValueType (page 235). Gets which type of value is found in the field.** | **Read only** | **None** |

### Remarks

- Regarding property EditMask, please see the additional information available in Seagate 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 MinimumValue 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), Page 170
AddCurrentValue Method (ParameterFieldDefinition Object), Page 170
AddDefaultValue Method (ParameterFieldDefinition Object), Page 171
ClearCurrentValueAndRange Method (ParameterFieldDefinition Object), Page 171
DeleteNthDefaultValue Method (ParameterFieldDefinition Object), Page 171
GetNthCurrentRange Method (ParameterFieldDefinition Object), Page 172
GetNthCurrentValue Method (ParameterFieldDefinition Object), Page 172
GetNthDefaultValue Method (ParameterFieldDefinition Object), Page 172
SetNthDefaultValue Method (ParameterFieldDefinition Object), Page 173

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>Sets start of the value range.</td>
</tr>
<tr>
<td>end</td>
<td>Sets end of the value range.</td>
</tr>
<tr>
<td>rangeInfo</td>
<td>CRRangeInfo (page 248). Use this bitwise value to indicate whether the upper and/or lower bound of the range should be included.</td>
</tr>
</tbody>
</table>

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 )
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DefaultValue</td>
<td>Specifies the default value to be added.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>Index of the value to be deleted.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>Index of the value range to be retrieved.</td>
</tr>
<tr>
<td>pStart</td>
<td>Sets start of the value range.</td>
</tr>
<tr>
<td>pEnd</td>
<td>Sets end of the value range.</td>
</tr>
<tr>
<td>pRangeInfo</td>
<td>CRRangeInfo (page 248). Use this bitwise value to indicate whether the upper and/or lower bound of the range should be included.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>Index number of the current value to be retrieved.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>The index number of the default value to be retrieved.</td>
</tr>
</tbody>
</table>

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

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>The index number of the default value to be set.</td>
</tr>
<tr>
<td>nthDefaultValue</td>
<td>Specifies the default value that you want to set for parameter field.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Long. Gets the number of parameter fields in the collection.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Item ( index As Long )</td>
<td><strong>ParameterFieldDefinition Object</strong> (page 166). Gets the item in the Collection specified by parameter index.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Parent</td>
<td><strong>Report Object</strong> (page 175). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
</tbody>
</table>
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), Page 174
Delete Method (ParameterFieldDefinitions Collection), Page 174

Add Method (ParameterFieldDefinitions Collection)

Use Add method to add a ParameterFieldDefinition Object (page 166), to the ParameterFieldDefinitions Collection.

Syntax

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameterName</td>
<td>Specifies the parameter name.</td>
</tr>
<tr>
<td>ValueType</td>
<td>CRFieldValueType (page 235). Specifies the value type of the field.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>Specifies the index number of the item that you want to delete from the Collection.</td>
</tr>
</tbody>
</table>
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 (page 175).

PrintingStatus Object Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumberOfPages</td>
<td>Long. Gets the total number of pages in the report.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>NumberOfRecordPrinted</td>
<td>Long. Gets the number of records printed.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>NumberOfRecordRead</td>
<td>Long. Gets the number of records read.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>NumberOfRecordSelected</td>
<td>Long. Gets the number of records selected.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Parent</td>
<td>Report Object (page 175). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Progress</td>
<td>CRPrintingProgress (page 247). Gets the printing progress.</td>
<td>Read only</td>
<td>None</td>
</tr>
</tbody>
</table>

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 (page 92). 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

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Application Object (page 92). Gets reference to the Application Object that the Report Object is associated with.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Access</td>
<td>Returned Type</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>ApplicationName</td>
<td>String. Gets or sets the application name.</td>
<td>Read/Write</td>
<td>None</td>
</tr>
</tbody>
</table>
| Areas                                        | **Areas Collection** (page 100). 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”, “GFI”.
For Example: RH, PH, GH1,...GHn, D, GFn,...GFI, RF, PF. The abbreviations for areas are case sensitive. | Read only   | None            |
<p>| BottomMargin                                 | Long. Gets or sets the page bottom margin, in twips.                                                                                                                                                           | Read/Write  | Can be written only when formatting idle. |
| CanPerformGroupingOnServer                   | Boolean. Gets whether the report can perform grouping on the server.                                                                                                                                           | Read only   | None            |
| 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. |
| ConvertDateTime Type                         | <strong>CRunconvertDateTimeType</strong> (page 231). 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. |
| ConvertNullField ToDefault                  | 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                                     | <strong>Database Object</strong> (page 109). 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. |
| EnableAsyncQuery                             | 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. |</p>
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Access</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnableParameter Prompting</td>
<td>Boolean. Gets or sets the prompting for parameter fields option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>EnablePerform Queries</td>
<td>Boolean. Gets or sets the perform queries asynchronously option</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>EnableSelectDistinct Records</td>
<td>Boolean. Gets or sets the select distinct records option</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>ExportOptions</td>
<td><strong>ExportOptions Object</strong> (page 122). Gets reference to ExportOptions Object for the report.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>FieldMappingType</td>
<td><strong>CRFieldMappingType</strong> (page 235). Gets or sets the field mapping type.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>FormulaFields</td>
<td><strong>FormulaFieldDefinitions Collection</strong> (page 135). Gets reference to Collection of all the named FormulaFieldDefinitions defined in the Report.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>FormulaSyntax</td>
<td><strong>CRFormulaSyntax</strong> (page 236). Gets or sets report formula syntax.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>GroupNameFields</td>
<td><strong>GroupNameFieldDefinitions Collection</strong> (page 144). Gets reference to a collection of all the group name fields defined in the report.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>GroupSelection Formula</td>
<td>String. Gets or sets the group selection formula.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>GroupSortFields</td>
<td><strong>SortFields Collection</strong> (page 208). Gets reference to group sort field collection.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>HasSavedData</td>
<td>Boolean. Gets whether the report has data saved in memory.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>KeywordsInReport</td>
<td>String. Gets or sets the keywords in the report.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>Kind</td>
<td><strong>CRReportKind</strong> (page 248). Gets what kind of report.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>LastGetFormula Syntax</td>
<td><strong>CRFormulaSyntax</strong> (page 236). Gets the formula syntax of the last formula text returned.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>LeftMargin</td>
<td>Long. Gets or sets the page left margin, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Access</td>
<td>Notes</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>MorePrintEngine</td>
<td>Boolean. Gets or sets the report option that indicates whether to pop up database error dialogs during printing when a Report Engine error occurs.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>ErrorMessages</td>
<td></td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>NumberOfGroups</td>
<td>Long. Gets the number of groups in the report.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>PageEngine</td>
<td>PageEngine Object (page 156). Gets reference to the PageEngine object.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>PaperOrientation</td>
<td>CRPaperOrientation (page 244). Gets or sets the current printer paper orientation. For the default printer, crDefaultPaperOrientation is returned.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>PaperSize</td>
<td>CRPaperSize (page 244). Gets or sets the current printer paper size. For the default printer, crDefaultPaperSize is returned.</td>
<td>Read/Write</td>
<td>Can be read or written only when formatting idle.</td>
</tr>
<tr>
<td>PaperSource</td>
<td>CRPaperSource (page 246). Gets or sets the current printer paper source.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>ParameterFields</td>
<td>ParameterFieldDefinitions Collection (page 173). 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).</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Parent</td>
<td>Report Object (page 175). Gets reference to the parent object for subreports. (NULL for main report).</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>PerformGroupingOnServer</td>
<td>Boolean. Gets or sets the performing grouping on server option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>PortName</td>
<td>String. Gets the printer port name used by the current report. Gets an empty string if the default printer is used.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
<td>Access Mode</td>
<td>Availability</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>PrintDate</td>
<td>Date. Gets or sets the print date for the report. By default, the current date will be used.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>PrinterDuplex</td>
<td><strong>CRPrinterDuplexType</strong> (page 247). Gets or sets the current printer duplex option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>PrinterName</td>
<td>String. Gets the printer name used by the report. Gets an empty string if the default printer is used.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>PrintingStatus</td>
<td><strong>PrintingStatus Object</strong> (page 175). Gets PrintingStatus Object for the report.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>RecordSelectionFormula</td>
<td>String. Gets or sets record selection formula.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>RecordSortFields</td>
<td><strong>SortFields Collection</strong> (page 208). Gets a collection of record sort fields.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>ReportAuthor</td>
<td>String. Gets or sets the report author.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>ReportComments</td>
<td>String. Gets or sets report comments.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>ReportSubject</td>
<td>String. Gets or sets the report subject.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>ReportTemplate</td>
<td>String. Gets or sets the report template.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>ReportTitle</td>
<td>String. Gets or sets the report title.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>RightMargin</td>
<td>Long. Gets or sets the page right margin, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>RunningTotalFields</td>
<td><strong>RunningTotalFieldDefinitions Collection</strong> (page 193). Gets running total fields collection.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>SavePreviewPicture</td>
<td>Boolean. Gets or sets save preview picture with report option.</td>
<td>Read/Write</td>
<td>None</td>
</tr>
<tr>
<td>Sections</td>
<td><strong>Sections Collection</strong> (page 206). Gets collection of all the sections in the report.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>SQLExpressionFields</td>
<td><strong>SQLExpressionFieldDefinitions Collection</strong> (page 212). Gets SQL expression field collection.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>SQLOpenQueryString</td>
<td>String. Gets or sets SQL query string.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
</tbody>
</table>
### SummaryFields

**SummaryFieldDefinitions Collection**

(page 220). Gets collection for group and report summaries (cross-tab summaries not available using this property).

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Access</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>TopMargin</td>
<td>Long. Gets or sets the page top margin, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>TranslateDosMemos</td>
<td>Boolean. Gets or sets the report option that indicates whether to translate DOS memos.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>TranslateDosStrings</td>
<td>Boolean. Gets or sets the report option that indicates whether to translate DOS strings.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>UseIndexForSpeed</td>
<td>Boolean. Gets or sets the use index for speed during record selection report option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>VerifyOnEveryPrint</td>
<td>Boolean. Gets or sets the report option that indicates whether to verify the database every time the report is printed.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
</tbody>
</table>

### Report Object Methods

The following methods are discussed in this section:

- **AddGroup Method (Report Object), Page 181**
- **AddReportVariable Method (Report Object), Page 181**
- **AutoSetUnboundFieldSource Method (Report Object), Page 182**
- **CancelPrinting Method (Report Object), Page 182**
- **DeleteGroup Method (Report Object), Page 182**
- **DiscardSavedData Method (Report Object), Page 183**
- **Export Method (Report Object), Page 183**
- **GetNextRows Method (Report Object), Page 183**
- **GetReportVariableValue Method (Report Object), Page 184**
- **OpenSubreport Method (Report Object), Page 184**
- **PrinterSetup Method (Report Object), Page 185**
- **PrintOut Method (Report Object), Page 185**
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupNumber</td>
<td>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.</td>
</tr>
<tr>
<td>pConditionField</td>
<td>Specifies the field to be grouped as the database field definition object.</td>
</tr>
<tr>
<td>Condition</td>
<td>CRGroupCondition (page 239). Specifies CRGroupCondition (see table below) indicating the grouping condition (that is, group on any value).</td>
</tr>
<tr>
<td>SortDirection</td>
<td>CRSortDirection (page 251). Specifies the sort direction for the group.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>CRReportVariableValueType (page 249). Specifies the type of variable that you want to add to the report.</td>
</tr>
<tr>
<td>pName</td>
<td>Specifies the unique name for the report variable that you want to add.</td>
</tr>
<tr>
<td>[arraySize As Long]</td>
<td>Reserved. Do not use.</td>
</tr>
<tr>
<td>[reserved]</td>
<td>Reserved. Do not use.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>matchType</td>
<td>CRBindingMatchType (page 230). Specifies whether to match name alone or name and data type.</td>
</tr>
<tr>
<td>[bindSubReports]</td>
<td>Specifies whether or not to bind subreport unbound fields.</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupNumber</td>
<td>Specifies the index number of the group that you want to delete from the report.</td>
</tr>
</tbody>
</table>

**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**

```vba
Sub DiscardSavedData ()
```

**Export Method (Report Object)**

Use Export method to export reports to a format and destination specified with **ExportOptions Object** (page 122).

**Syntax**

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[promptUser]</td>
<td>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).</td>
</tr>
</tbody>
</table>

**GetNextRows Method (Report Object)**

Use GetNextRows method to get the specified rowset.

**Syntax**

```vba
Function GetNextRows ( startRowN As Long, pRowN As Long )
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>startRowN</td>
<td>Specifies the row number to start the rowset on.</td>
</tr>
<tr>
<td>pRowN</td>
<td>Specifies the number of rows to return to the rowset.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pName</td>
<td>Specifies the unique name of the report variable for which you want to get the value.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pSubreportName</td>
<td>Specifies the file name of the subreport to be opened.</td>
</tr>
</tbody>
</table>
**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**
```vba
Sub PrinterSetup ( hWnd As Long )
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hWnd</td>
<td>Specifies the handle of the printer setup dialog box parent window. If you pass 0, the top level application window will be the parent.</td>
</tr>
</tbody>
</table>

**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)`, Page 186. 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**
```vba
Sub PrintOut ( [promptUser], [numberOfCopy], [collated], [startPageN], [stopPageN] )
```

**Parameters**

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pFilePath</td>
<td>Specifies the file path and name that you want to use to save the report.</td>
</tr>
<tr>
<td>fileFormat</td>
<td>CRReportFileFormat (page 248). Specifies the file format that you want to use to save the report.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pDriverName</td>
<td>Specifies the name of the printer driver for the selected printer.</td>
</tr>
<tr>
<td>pPrinterName</td>
<td>Specifies the printer name for the selected printer.</td>
</tr>
<tr>
<td>pPortName</td>
<td>Specifies the port name for the port to which the selected printer is attached.</td>
</tr>
</tbody>
</table>
SetDialogParentWindow Method (Report Object)
The SetDialogParentWindow method sets the dialog parent window.

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

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hWnd</td>
<td>Specifies the handle of the parent window.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pName</td>
<td>Specifies the name of the variable for which you want to set the value.</td>
</tr>
<tr>
<td>var</td>
<td>Specifies the value that you want to set.</td>
</tr>
</tbody>
</table>

Report Object Events
The following events are discussed in this section:
AfterFormatPage Event (Report Object), Page 188
BeforeFormatPage Event (Report Object), Page 188
FieldMapping Event (Report Object), Page 188
NoData Event (Report Object), Page 189
**AfterFormatPage Event (Report Object)**
The AfterFormatPage event is fired after formatting a page.

**Syntax**
```vbnet
Event AfterFormatPage ( PageNumber As Long )
```

**Parameter**
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PageNumber</td>
<td>Specifies the number of the report page triggering the event.</td>
</tr>
</tbody>
</table>

**BeforeFormatPage Event (Report Object)**
The BeforeFormatPage event is fired before formatting a page.

**Syntax**
```vbnet
Event BeforeFormatPage ( PageNumber As Long )
```

**Parameter**
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PageNumber</td>
<td>Specifies the number of the report page triggering the event.</td>
</tr>
</tbody>
</table>

**FieldMapping Event (Report Object)**
The FieldMapping event fires if the database is changed while verifying database.

**Syntax**
```vbnet
Event FieldMapping ( reportFieldArray, databaseFieldArray, useDefault As Boolean )
```

**Parameters**
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reportFieldArray</td>
<td>Specifies the report field array to map.</td>
</tr>
<tr>
<td>databaseFieldArray</td>
<td>Specifies the database field array to map.</td>
</tr>
<tr>
<td>useDefault</td>
<td>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.</td>
</tr>
</tbody>
</table>
NoData Event (Report Object)
The NoData event fires when there is no data for the report.

Syntax

\[
\text{Event NoData ( pCancel As Boolean )}
\]

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pCancel</td>
<td>Specifies whether to cancel the report.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Long. Gets the number of report objects in the collection.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Item ( index ) As Object</td>
<td>Gets a report object. Depending on the type of item referenced, this can be a BlobFieldObject Object (page 101), FieldObject Object (page 128), TextObject Object (page 224), Page Object (page 154), CrossTabObject Object (page 106), SubreportLinks Collection (page 214), BoxObject Object (page 103), GraphObject Object (page 137), LineObject Object (page 146), MapObject Object (page 147), OlapGridObject Object (page 150). 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.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Parent</td>
<td>Section Object (page 195). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
</tbody>
</table>

Remarks

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

<table>
<thead>
<tr>
<th>Property</th>
<th>Definition</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluate Condition</td>
<td>CRRunningTotalCondition (page 250). Gets evaluate condition.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Evaluate ConditionField</td>
<td>Object. Gets evaluate condition field.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Evaluate ConditionFormula</td>
<td>String. Gets or sets evaluate condition formula.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>Evaluate GroupNumber</td>
<td>Integer. Gets or sets evaluate group number.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>Kind</td>
<td>CRFieldKind (page 235). Gets which kind of field (that is, database, summary, formula, etc.).</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Name</td>
<td>String. Gets the unique formula name of the field within the report (table.FIELD). For example, [#Test].</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>NextValue</td>
<td>Variant. Gets the field next value.</td>
<td>Read only</td>
<td>Can be read only when top-level Report object is formatting active.</td>
</tr>
<tr>
<td>NumberOfBytes</td>
<td>Integer. Gets the number of bytes required to store the field data in memory.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Parent</td>
<td>Report Object (page 175). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>PreviousValue</td>
<td>Variant. Gets the field previous value.</td>
<td>Read only</td>
<td>Can be read only when top-level Report object is formatting active.</td>
</tr>
</tbody>
</table>
### RunningTotalFieldDefinition Object Methods

The following methods are discussed in this section:

- `SetEvaluateConditionField Method (RunningTotalFieldDefinition Object), Page 192`
- `SetNoEvaluateCondition Method (RunningTotalFieldDefinition Object), Page 192`
- `SetNoResetCondition Method (RunningTotalFieldDefinition Object), Page 192`
- `SetResetConditionField Method (RunningTotalFieldDefinition Object), Page 192`
- `SetSecondarySummarizedField Method (RunningTotalFieldDefinition Object), Page 193`
- `SetSummarizedField Method (RunningTotalFieldDefinition Object), Page 193`

<table>
<thead>
<tr>
<th>Property</th>
<th>Definition</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResetCondition</td>
<td><code>CRRunningTotalCondition</code> (page 250). Gets the reset condition.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>ResetConditionField</td>
<td>Object. Gets reset condition field.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>ResetConditionFormula</td>
<td>String. Gets or sets the reset condition formula.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>ResetGroupNumber</td>
<td>Integer. Gets or sets the reset group number.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>RunningTotalFieldName</td>
<td>String. Gets the running total field name.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>SecondarySummarizedField</td>
<td>Object. Gets the secondary summarized field.</td>
<td>Read only</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>SummarizedField</td>
<td>Object. Gets the summarized field.</td>
<td>Read only</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>SummaryOperationParameter</td>
<td>Long. Gets or sets summary operation parameter.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>SummaryType</td>
<td><code>CRSummaryType</code> (page 251). Gets or sets summary type.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>Value</td>
<td>Variant. Gets the field current value.</td>
<td>Read only</td>
<td>Can be read only when top-level Report object is formatting active.</td>
</tr>
<tr>
<td>ValueType</td>
<td><code>CRFieldValueType</code> (page 235). Gets which type of value is found in the field.</td>
<td>Read only</td>
<td>None</td>
</tr>
</tbody>
</table>
**SetEvaluateConditionField Method**  
**RunningTotalFieldDefinition Object**  
Use SetEvaluateConditionField to set the evaluate condition field.

**Syntax**

```vba
Sub SetEvaluateConditionField ( pEvaluateConditionField )
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pEvaluateConditionField</td>
<td>Specifies the condition field that you want to use.</td>
</tr>
</tbody>
</table>

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

**Syntax**

```vba
Sub SetNoEvaluateCondition ()
```

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

**Syntax**

```vba
Sub SetNoResetCondition ()
```

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

**Syntax**

```vba
Sub SetResetConditionField ( pResetConditionField )
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pResetConditionField</td>
<td>Specifies the condition field that you want to use.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>secondarySummariedField</td>
<td>Specifies the condition field that you want to use.</td>
</tr>
</tbody>
</table>

SetSummarizedField Method (RunningTotalFieldDefinition Object)

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

Syntax

Sub SetSummarizedField ( SummarizedField )

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SummarizedField</td>
<td>Specifies the condition field that you want to use.</td>
</tr>
</tbody>
</table>

RunningTotalFieldDefinitions Collection

The RunningTotalFieldDefinitions Collection is a collection of running total field definition objects. One object exists in the collection for every running total field accessed by the report. Access a specific RunningTotalFieldDefinition Object (page 190), in the collection using the Item property.
RunningTotalFieldDefinitions Collection Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Long. Gets the number of RunningTotalFieldDefinition Object (page 190), in the collection.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Item (index As Long)</td>
<td>RunningTotalFieldDefinition Object (page 190). 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.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Parent</td>
<td>Report Object (page 175). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
</tbody>
</table>

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), Page 194
Delete Method (RunningTotalFieldDefinitions Collection), Page 195

Add Method (RunningTotalFieldDefinitions Collection)
Use Add method to add a RunningTotalFieldDefinition Object (page 190) to the Collection.

Syntax

Function Add (runningTotalName As String ) As RunningTotalFieldDefinition

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>runningTotalName</td>
<td>Specifies the name of the RunningTotal field that you want to add.</td>
</tr>
</tbody>
</table>

Returns
Returns a RunningTotalFieldDefinition Object (page 190) member of the Collection.
Delete Method (RunningTotalFieldDefinitions Collection)

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

**Syntax**

```vba
Sub Delete ( index )
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>Specifies the index number of the Object that you want to remove from the Collection.</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackColor</td>
<td>OLE_COLOR. Gets or sets the section background color.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Height</td>
<td>Long. Gets or sets the height of the section, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>KeepTogether</td>
<td>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.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>MinimumHeight</td>
<td>Long. Gets the minimum section height, in twips.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Name</td>
<td>String. Gets or sets the name of the section.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>NewPageAfter</td>
<td>Boolean. Gets or sets the option that indicates whether to start a new page after the current section.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
</tbody>
</table>
### Section Object Methods

The following methods are discussed in this section:

AddBlobFieldObject Method (Section Object), Page 197
AddBoxObject Method (Section Object), Page 197

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewPageBefore</td>
<td>Boolean. Gets or sets the option that indicates whether to start a new page before the current section.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Number</td>
<td>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).</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Parent</td>
<td>Area Object (page 98). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>PrintAtBottomOfPage</td>
<td>Boolean. Gets or sets the option that indicates whether to print the current section at the bottom of the page.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>ReportObjects</td>
<td>ReportObjects Collection (page 189). Gets reference to the heterogeneous collection of report objects for the section.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>ResetPageNumberAfter</td>
<td>Boolean. Gets or sets the option that indicates whether to reset the page number after the current section.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Suppress</td>
<td>Boolean. Gets or sets the section visibility.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>SuppressIfBlank</td>
<td>Boolean. Gets or sets the option that indicates whether to suppress the current section if it is blank.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>UnderlaySection</td>
<td>Boolean. Gets or sets the underlay following section option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Width</td>
<td>Long. Gets the width of the section, in twips.</td>
<td>Read only</td>
<td>None</td>
</tr>
</tbody>
</table>
AddBlobFieldObject Method (Section Object)
The AddBlobFieldObject method adds a BlobFieldObject Object (page 101) to the Section Object.

Syntax

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>Variant. Can be formula form name or any field definition that specifies the field that you want to add.</td>
</tr>
<tr>
<td>Left</td>
<td>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.</td>
</tr>
<tr>
<td>Top</td>
<td>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.</td>
</tr>
</tbody>
</table>

Returns

Returns a BlobFieldObject Object (page 101).

AddBoxObject Method (Section Object)
The BoxObject method adds a BoxObject Object (page 103) to the Section Object.
Syntax

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>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.</td>
</tr>
<tr>
<td>Top</td>
<td>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.</td>
</tr>
<tr>
<td>Right</td>
<td>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.</td>
</tr>
<tr>
<td>Bottom</td>
<td>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.</td>
</tr>
<tr>
<td>[pEndSection]</td>
<td>Specifies the Section in which the end of the BoxObject will be placed, if not the parent Section.</td>
</tr>
</tbody>
</table>

Returns

Returns a **BoxObject Object** (page 103).

**AddCrossTabObject Method (Section Object)**

The AddCrossTabObject method adds a **CrossTabObject Object** (page 106) 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>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.</td>
</tr>
<tr>
<td>Top</td>
<td>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.</td>
</tr>
</tbody>
</table>

Returns

Returns a **CrossTabObject Object** (page 106).
AddFieldObject Method (Section Object)

The AddFieldObject method adds a FieldObject Object (page 128) 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>Specifies the field that you want to add to the Section.</td>
</tr>
<tr>
<td>Left</td>
<td>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.</td>
</tr>
<tr>
<td>Top</td>
<td>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.</td>
</tr>
</tbody>
</table>

Returns

Returns a FieldObject Object (page 128).

AddGraphObject Method (Section Object)

The AddGraphObject method adds a GraphObject Object (page 137) (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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>graphDataType</td>
<td>CRGraphDataType (page 237). Specifies the data type for the graph that you want to add.</td>
</tr>
<tr>
<td>Left</td>
<td>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.</td>
</tr>
<tr>
<td>Top</td>
<td>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.</td>
</tr>
</tbody>
</table>
Returns
Returns a **GraphObject Object** (page 137).

**AddLineObject Method (Section Object)**
The AddLineObject method adds a **LineObject Object** (page 146) to the Section Object. If optional parameter [pEndSection] is passed different from the current section, you can add a vertical line across sections.

**Syntax**

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>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.</td>
</tr>
<tr>
<td>Top</td>
<td>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.</td>
</tr>
<tr>
<td>Right</td>
<td>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.</td>
</tr>
<tr>
<td>Bottom</td>
<td>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.</td>
</tr>
<tr>
<td>[pEndSection]</td>
<td>Specifies the Section in which the end of the LineObject will be placed, if not the parent Section.</td>
</tr>
</tbody>
</table>

Returns
Returns a **LineObject Object** (page 146).

**AddPictureObject Method (Section Object)**
The AddLineObject method adds a picture object from an image in the form of an **OleObject Object** (page 151) to the Section Object.

**Syntax**

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pImageFilePath</td>
<td>Specifies the image file path and name that you want to add.</td>
</tr>
<tr>
<td>Left</td>
<td>Specifies the offset of the top left corner of the Object that you are adding</td>
</tr>
<tr>
<td></td>
<td>relative to the top left corner of the parent Section, in twips.</td>
</tr>
<tr>
<td>Top</td>
<td>Specifies the offset of the top left corner of the Object that you are adding</td>
</tr>
<tr>
<td></td>
<td>relative to the top left corner of the parent Section, in twips.</td>
</tr>
</tbody>
</table>

Returns

Returns a **OleObject Object** (page 151).

**AddSpecialVarFieldObject Method (Section Object)**

The SpecialVarFieldObject Method adds a SpecialVar **FieldObject Object** (page 128) to the Section Object.

Syntax

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>specialVarType</td>
<td><strong>CRSpecialVarType</strong> (page 251). Specifies the SpecialVar type.</td>
</tr>
<tr>
<td>Left</td>
<td>Specifies the offset of the top left corner of the Object that you are adding</td>
</tr>
<tr>
<td></td>
<td>relative to the top left corner of the parent Section, in twips.</td>
</tr>
<tr>
<td>Top</td>
<td>Specifies the offset of the top left corner of the Object that you are adding</td>
</tr>
<tr>
<td></td>
<td>relative to the top left corner of the parent Section, in twips.</td>
</tr>
</tbody>
</table>

Returns

Returns a **FieldObject Object** (page 128).

**AddSubreportObject Method (Section Object)**

The SubReportObject method adds a **SubreportObject Object** (page 216) to the Section Object. This method adds an empty subreport to the Section. You can then add Objects and Sections to the SubreportObject.

Syntax

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pSubreportName</td>
<td>Specifies the name of the subreport that you want to add.</td>
</tr>
<tr>
<td>Left</td>
<td>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.</td>
</tr>
<tr>
<td>Top</td>
<td>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.</td>
</tr>
</tbody>
</table>

Returns

Returns a SubreportObject Object (page 216).

AddSummaryFieldObject Method (Section Object)

The AddSummaryField method adds a summary FieldObject Object (page 128) to the Section Object.

Syntax

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>Specifies the summary field that you want to add.</td>
</tr>
<tr>
<td>SummaryType</td>
<td>CRSummaryType (page 251). Specifies the type of summary field.</td>
</tr>
<tr>
<td>Left</td>
<td>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.</td>
</tr>
<tr>
<td>Top</td>
<td>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.</td>
</tr>
<tr>
<td>[secondSummaryFieldOrFactor]</td>
<td>Specifies the second summary field or factor, if required by the summary type.</td>
</tr>
</tbody>
</table>

Returns

Returns a FieldObject Object (page 128).

AddTextObject Method (Section Object)

The AddTextObject method adds a TextObject Object (page 224) 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pText</td>
<td>Specifies the text that you want to add.</td>
</tr>
<tr>
<td>Left</td>
<td>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.</td>
</tr>
<tr>
<td>Top</td>
<td>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.</td>
</tr>
<tr>
<td>[formatText]</td>
<td>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.</td>
</tr>
</tbody>
</table>

Returns

Returns a FieldObject Object (page 128).

AddUnboundFieldObject Method (Section Object)

The AddUnboundFieldObject method adds an unbound FieldObject Object (page 128) 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ValueType</td>
<td>CRFieldValueType (page 235). Specifies the type of value in the field.</td>
</tr>
<tr>
<td>Left</td>
<td>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.</td>
</tr>
<tr>
<td>Top</td>
<td>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.</td>
</tr>
</tbody>
</table>

Returns

Returns a FieldObject Object (page 128).
DeleteObject Method (Section Object)

The DeleteObject method removes an object from the Section Object

Syntax

Sub DeleteObject ( reportObject )

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reportObject</td>
<td>Specifies the report object that you want to remove from the Section.</td>
</tr>
</tbody>
</table>

ImportSubreport Method (Section Object)

The ImportSubreport method imports a SubreportObject Object (page 216) 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subreportFileName</td>
<td>Specifies the name of the subreport that you want to import.</td>
</tr>
<tr>
<td>Left</td>
<td>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.</td>
</tr>
<tr>
<td>Top</td>
<td>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.</td>
</tr>
</tbody>
</table>

Returns

Returns a SubreportObject Object (page 216).

Section Object Events

The following events are discussed in this section:

Format Event (Section Object), Page 205
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

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

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FormattingInfo</td>
<td>The FormattingInfo object which will contain formatting information to be used by the Format event handler.</td>
</tr>
</tbody>
</table>

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:

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

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 (page 175) or Area Object (page 98). 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

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Long. Gets the number of section in the collection.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Item (index)</td>
<td>Section Object (page 195). 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.).</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Parent</td>
<td>Report Object (page 175) or Area Object (page 98). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
</tbody>
</table>
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), Page 207
Delete Method (Sections Collection), Page 207

Add Method (Sections Collection)
Use Add method to add a Section Object (page 195) to the Sections Collection.

Syntax
Function Add ( [index] ) As Section

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[index]</td>
<td>Specifies the index where you would like to add the Section to the Collection.</td>
</tr>
</tbody>
</table>

Returns
Returns a Section Object (page 195) member of the Sections Collection.

Delete Method (Sections Collection)
Use Delete method to remove a Section Object (page 195) from the Sections Collection.

Syntax
Sub Delete(index)

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>Specifies the index of the Section that you want to delete from the Collection.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>Object. Gets or sets the sorting field definition object.</td>
<td>Read/Write</td>
<td>Can be read or written only when formatting idle.</td>
</tr>
<tr>
<td>Parent</td>
<td>Report Object (page 175). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>SortDirection</td>
<td>CRSortDirection (page 251). Gets or sets the sort direction.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Long. Gets the number of sort fields in the collection.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Item (index As Long)</td>
<td>SortField Object (page 208). 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.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Parent</td>
<td>Report Object (page 175). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
</tbody>
</table>
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), Page 209
Delete Method (SortFields Collection), Page 209

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pFieldDefinition</td>
<td>IFieldDefinition Object (page 144). Specifies the field definition or field name.</td>
</tr>
<tr>
<td>SortDirection</td>
<td>CRSortDirection (page 251). Specifies the direction in which the field data should be sorted (that is, ascending, descending, etc.).</td>
</tr>
</tbody>
</table>

Delete Method (SortFields Collection)
The Delete method deletes a record or group sort field from the Collection.

Syntax
Sub Delete ( index As Long )

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>Specifies the 1-based index number of the sort field in the collection that you want to delete.</td>
</tr>
</tbody>
</table>
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 (page 128).

**SpecialVarFieldDefinition Object Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kind</td>
<td><strong>CRFieldKind</strong> (page 235). Gets which kind of field (database, summary, formula, etc.).</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Name</td>
<td>String. Gets the field definition unique formula name of the special var. field.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>NextValue</td>
<td>Variant. Gets the field next value.</td>
<td>Read only</td>
<td>Can be read only when top-level Report object is formatting active.</td>
</tr>
<tr>
<td>NumberOfBytes</td>
<td>Integer. Gets the number of bytes required to store the field data in memory.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Parent</td>
<td><strong>Report Object</strong> (page 175). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>PreviousValue</td>
<td>Variant. Gets the field previous value.</td>
<td>Read only</td>
<td>Can be read only when top-level Report object is formatting active.</td>
</tr>
<tr>
<td>SpecialVarType</td>
<td><strong>CRSpecialVarType</strong> (page 251). Gets what the type of special field (for example, ReportTitle, PageNumber, etc.).</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Value</td>
<td>Variant. Gets the field current value.</td>
<td>Read only</td>
<td>Can be read only when top-level Report object is formatting active.</td>
</tr>
<tr>
<td>ValueType</td>
<td><strong>CRFieldValueType</strong> (page 235). Gets which type of value is found in the field.</td>
<td>Read only</td>
<td>None</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Property</th>
<th>Definition</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kind</td>
<td><strong>CRFieldKind</strong> (page 235). Gets which kind of field (that is, database, summary, formula, etc.).</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Name</td>
<td>String. Gets the field definition unique formula name of the field within the report, Crystal formula syntax.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>NextValue</td>
<td>Variant. Gets the field next value.</td>
<td>Read only</td>
<td>Can be read only when top-level Report object is formatting active.</td>
</tr>
<tr>
<td>NumberOfBytes</td>
<td>Integer. Gets the number of bytes required to store the field data in memory.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Parent</td>
<td><strong>Report Object</strong> (page 175). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>PreviousValue</td>
<td>Variant. Gets the field previous value.</td>
<td>Read only</td>
<td>Can be read only when top-level Report object is formatting active.</td>
</tr>
<tr>
<td>SQLExpressionFieldName</td>
<td>String. Gets the SQL expression field name.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Text</td>
<td>String. Gets or sets the SQL expression text.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>Value</td>
<td>Variant. Gets the field current value.</td>
<td>Read only</td>
<td>Can be read only when top-level Report object is formatting active.</td>
</tr>
<tr>
<td>ValueType</td>
<td><strong>CRFieldValueType</strong> (page 235). Gets which type of value is found in the field.</td>
<td>Read only</td>
<td>None</td>
</tr>
</tbody>
</table>
SQLExpressionFieldDefinition Object Methods

The following methods are discussed in this section:

Check Method (SQLExpressionFieldDefinition Object), Page 212

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pBool</td>
<td>Boolean value indicating the condition of the formula string. Will be set to</td>
</tr>
<tr>
<td></td>
<td>TRUE if the formula is valid and FALSE if the formula contains one or more</td>
</tr>
<tr>
<td></td>
<td>errors.</td>
</tr>
<tr>
<td>ppErrorString</td>
<td>Specifies the error message string if the formula contains an error.</td>
</tr>
</tbody>
</table>

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 (page 211) in the collection using the Item property.

SQLExpressionFieldDefinitions Collection Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Long. Gets the number of items in the Collection.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Item ( index As Long )</td>
<td>SQLExpressionFieldDefinition Object (page 211). 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.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Parent</td>
<td>Report Object (page 175). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
</tbody>
</table>
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), Page 213
Delete Method (SQLExpressionFieldDefinitions Collection), Page 213

Add Method (SQLExpressionFieldDefinitions Collection)
Use Add method to add a SQLExpressionFieldDefinition Object (page 211) to the Collection.

Syntax

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLExpressionName</td>
<td>Specifies the name of the SQL expression that you want to add to the Collection.</td>
</tr>
<tr>
<td>Text</td>
<td>Specifies the text of the SQL expression that you want to add to the Collection.</td>
</tr>
</tbody>
</table>

Returns

Returns a SQLExpressionFieldDefinition Object (page 211) member of the Collection.

Delete Method (SQLExpressionFieldDefinitions Collection)
Use Delete method to remove a SQLExpressionFieldDefinition Object (page 211) from the Collection.

Syntax

```
Sub Delete ( index )
```
SubreportLink Object

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

SubreportLink Object Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>MainReportField</td>
<td><code>IFieldDefinition Object</code> (page 144). Gets the main report field to which the subreport field is linked.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Parent</td>
<td><code>SubreportObject Object</code> (page 216). Gets the subreport link’s parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>SubReportField</td>
<td><code>IFieldDefinition Object</code> (page 144). Gets the subreport field.</td>
<td>Read only</td>
<td>None</td>
</tr>
</tbody>
</table>

SubreportLinks Collection

The SubreportLinks Collection contains the SubreportLink Object (page 214) related to the report.

SubreportLinks Collection Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Long. Gets a count of the items in the Collection.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Item ( index As Long )</td>
<td><code>SubreportLink Object</code> (page 214). Gets an item from the Collection.</td>
<td>Read only</td>
<td>None</td>
</tr>
</tbody>
</table>
SubreportLinks Collection Methods

The following methods are discussed in this section:

Add Method (SubreportLinks Collection), Page 215
Delete Method (SubreportLinks Collection), Page 215

**Add Method (SubreportLinks Collection)**

Use Add method to add a SubreportLink Object (page 214) to the Collection.

Syntax

```
Function Add ( MainReportField, SubreportField ) As SubreportLink
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MainReportField</td>
<td>Specifies the field in the main report that you want the new Object in the Collection to link.</td>
</tr>
<tr>
<td>SubreportField</td>
<td>Specifies the field in the subreport that you want the new Object in the Collection to link.</td>
</tr>
</tbody>
</table>

**Returns**

Returns a SubreportLink Object (page 214) member of the Collection.

**Delete Method (SubreportLinks Collection)**

Use Delete method to remove a SubreportLink Object (page 214) from the Collection.

Syntax

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>Specifies the index number of the Object in the Collection that you want to remove.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackColor</td>
<td>OLE_COLOR. Gets or sets the object background color.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>BorderColor</td>
<td>OLE_COLOR. Gets or sets the object border color.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>BottomLineStyle</td>
<td>CRLineStyle (page 241). Gets or sets the bottom line style.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>CanGrow</td>
<td>Boolean. Gets or sets the can grow option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>CloseAtPageBreak</td>
<td>Boolean. Gets or sets close border on page break.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>EnableOnDemand</td>
<td>Boolean. Gets the real-time subreport option.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>HasDropShadow</td>
<td>Boolean. Gets or sets the border drop shadow option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Height</td>
<td>Long. Gets or sets the object height, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>KeepTogether</td>
<td>Boolean. Gets or sets the keep object together option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Kind</td>
<td>CRObj ectKind (page 244). Gets what kind of object (for example, box, cross-tab, field).</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Left</td>
<td>Long. Gets or sets the object upper left position, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>LeftLineStyle</td>
<td>CRLineStyle (page 241). Gets or sets the left line style.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
</tbody>
</table>

Seagate Crystal Reports Technical Reference
SubreportObject Object Methods

The following methods are discussed in this section:

OpenSubreport Method (SubreportObject Object), Page 217

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 (page 216).

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Links</td>
<td>SubreportLinks Collection (page 214). Gets reference to the Collection.</td>
<td>Read Only</td>
<td>None</td>
</tr>
<tr>
<td>Name</td>
<td>String. Gets or sets the object name.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Parent</td>
<td>Section Object (page 195). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>RightLineStyle</td>
<td>CRLineStyle (page 241). Gets or sets the right line style.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>SubreportName</td>
<td>String. Gets or sets the name of the subreport.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>Suppress</td>
<td>Boolean. Gets or sets the object visibility option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Top</td>
<td>Long. Gets or sets the object upper top position, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>TopLineStyle</td>
<td>CRLineStyle (page 241). Gets or sets the top line style.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Width</td>
<td>Long. Gets or sets the object width, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
</tbody>
</table>
### SummaryFieldDefinition Object

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

#### SummaryFieldDefinition Object Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>FooterArea</td>
<td><strong>Area Object</strong> (page 98). Gets the area pair that the summary is in.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>ForCrossTab</td>
<td>Boolean. Gets for-cross-tab.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>HeaderArea</td>
<td><strong>Area Object</strong> (page 98). Gets the area pair that the summary is in.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Kind</td>
<td><strong>CRFieldKind</strong> (page 235). Gets which kind of field (database, summary, formula, etc.).</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Name</td>
<td>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.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>NextValue</td>
<td>Variant. Gets the field next value.</td>
<td>Read only</td>
<td>Can be read only when top-level Report object is formatting.</td>
</tr>
<tr>
<td>NumberOfBytes</td>
<td>Integer. Gets the number of bytes required to store the field data in memory.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Parent</td>
<td><strong>Report Object</strong> (page 175). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>PreviousValue</td>
<td>Variant. Gets the field previous value.</td>
<td>Read only</td>
<td>Can be read only when top-level Report object is formatting.</td>
</tr>
<tr>
<td>SecondarySummarizedField</td>
<td>Object. Gets the secondary summarized field.</td>
<td>Read only</td>
<td>None</td>
</tr>
</tbody>
</table>
SummaryFieldDefinition Object Methods

The following methods are discussed in this section:

- SetSecondarySummarizedField Method (SummaryFieldDefinition Object), Page 219
- SetSummarizedField Method (SummaryFieldDefinition Object), Page 220

### SetSecondarySummarizedField Method (SummaryFieldDefinition Object)

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

**Syntax**

```vba
Sub SetSecondarySummarizedField (secondarySummaredField)
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>secondarySummaredField</td>
<td>Specifies the field that you want to set.</td>
</tr>
</tbody>
</table>
SetSummarizedField Method (SummaryFieldDefinition Object)

Syntax

```vbnet
Sub SetSummarizedField ( SummarizedField )
```

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SummarizedField</td>
<td>Specifies the field that you want to set.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Long. Gets the number of summary fields in the collection.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Item (index As Long)</td>
<td><strong>SummaryFieldDefinition Object</strong> (page 218). 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.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Parent</td>
<td><strong>Report Object</strong> (page 175). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
</tbody>
</table>

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), Page 221
Delete Method (SummaryFieldDefinitions Collection), Page 221

Add Method (SummaryFieldDefinitions Collection)

Use Add method to add a SummaryFieldDefinition Object (page 218) to the Collection.

Syntax

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>groupLevel</td>
<td>Specifies the group level for the summary.</td>
</tr>
<tr>
<td>Field</td>
<td>Specifies the summary field.</td>
</tr>
<tr>
<td>SummaryType</td>
<td>CRSummaryType (page 251). Specifies the summary type.</td>
</tr>
<tr>
<td>[secondSummaryFieldOrFactor]</td>
<td>Specifies the second summary field or factor.</td>
</tr>
</tbody>
</table>

Returns

Returns a SummaryFieldDefinition Object (page 218) member of the Collection.

Delete Method (SummaryFieldDefinitions Collection)

Use delete method to delete a SummaryFieldDefinition Object (page 218) from the Collection.

Syntax

Sub Delete ( index )

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>Specifies the index number of the object that you want to delete from the Collection.</td>
</tr>
</tbody>
</table>
TableLink Object

The TableLink Object provides information about database table links.

TableLink Object Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>DestinationFields</td>
<td>DatabaseFieldDefinitions Collection (page 116). Gets reference to table link destination field definitions Collection.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>DestinationTable</td>
<td>Database Table Object (page 116). Gets reference to the table link destination table.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>IndexUsed</td>
<td>Integer. Gets table link IndexUsed</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>JoinType</td>
<td>CRLinkJoinType (page 241). Gets the table link join type.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>LookupType</td>
<td>CRLinkLookUpType (page 242). Gets table link lookup type.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Parent</td>
<td>Database Object (page 109). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>PartialMatch Enabled</td>
<td>Boolean. Gets the table link partial-match enabled option.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>SourceFields</td>
<td>DatabaseFieldDefinition Object (page 115). Gets the table link source field definitions.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>SourceTable</td>
<td>Database Table Object (page 116). Gets table link source table.</td>
<td>Read only</td>
<td>None</td>
</tr>
</tbody>
</table>
TableLinks Collection

The TableLinks Collection contains the **TableLink Object** (page 222) Objects associated with the report.

### TableLinks Collection Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Definition</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Long. Gets a count of the collection items.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Item (index As Long)</td>
<td><strong>TableLink Object</strong> (page 222). Gets an item from the Collection.</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Parent</td>
<td><strong>Database Object</strong> (page 109). Gets reference to the parent object.</td>
<td>Read only</td>
<td>None</td>
</tr>
</tbody>
</table>

### TableLinks Collection Methods

The following methods are discussed in this section:

- **Add Method (TableLinks Collection), Page 223**
- **Delete Method (TableLinks Collection), Page 224**

#### Add Method (TableLinks Collection)

Use Add method to add a **TableLink Object** (page 222) to the Collection.

**Syntax**

```vba
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>psrcTable</td>
<td><strong>DatabaseTable Object</strong> (page 116). Specifies the source database table.</td>
</tr>
<tr>
<td>pDestTable</td>
<td><strong>DatabaseTable Object</strong> (page 116). Specifies the destination database table.</td>
</tr>
</tbody>
</table>
Returns a `TableLink Object` (page 222) member of the Collection.

Delete Method (TableLinks Collection)

Use Delete method to remove a `TableLink Object` (page 222) from the Collection.

Syntax

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

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>index</code></td>
<td>Specifies the index number of the item that you want to remove from the Collection.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
<th>Restriction in event handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackColor</td>
<td>OLE_COLOR. Gets or sets the object background color.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>BorderColor</td>
<td>OLE_COLOR. Gets or sets the object border color.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>BottomLineStyle</td>
<td><code>CRLinestyle</code> (page 241). Gets or sets the bottom line style.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Read/Write</td>
<td>Restriction in event handler</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>CanGrow</td>
<td>Boolean. Gets or sets the can grow option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>CharacterSpacing</td>
<td>Long. Gets or sets the character spacing.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>CloseAtPageBreak</td>
<td>Boolean. Gets or sets the close border on page break option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>FirstLineIndent</td>
<td>Long. Gets or sets first line indent, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>Font</td>
<td>IFontDisp. Gets or sets the standard OLE font of the text object.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>HasDropShadow</td>
<td>Boolean. Gets or sets the border drop shadow option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Height</td>
<td>Long. Gets or sets the object height, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>HorAlignment</td>
<td>CRAIignment (page 229). Gets or sets the horizontal alignment.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>KeepTogether</td>
<td>Boolean. Gets or sets the keep area together option.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Kind</td>
<td>CRCObjectKind (page 244). Gets what kind of object (for example, box, cross-tab, field).</td>
<td>Read only</td>
<td>None</td>
</tr>
<tr>
<td>Left</td>
<td>Long. Gets or sets the object upper left position, in twips.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>LeftIndent</td>
<td>Long. Gets or sets left indent.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>LeftLineStyle</td>
<td>CRLineStyle (page 241). Gets or sets the left line style.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>LineSpacing</td>
<td>Double. Gets the line spacing.</td>
<td>Read only</td>
<td>Can be written only when formatting idle.</td>
</tr>
<tr>
<td>LineSpacingType</td>
<td>CRLineSpacingType (page 241). Gets the line spacing type.</td>
<td>Read only</td>
<td></td>
</tr>
<tr>
<td>MaxNumberOfLines</td>
<td>Integer. Gets or sets the maximum number of line for a string memo field.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle or active.</td>
</tr>
<tr>
<td>Name</td>
<td>String. Gets or sets area name.</td>
<td>Read/Write</td>
<td>Can be written only when formatting idle.</td>
</tr>
</tbody>
</table>
### TextObject Object Methods

The following methods are discussed in this section:

- SetLineSpacing Method (TextObject Object), Page 227
- SetLineSpacing Method (TextObject Object), Page 227
SetLineSpacing Method (TextObject Object)
Use the SetLineSpacing method to set the line spacing for a TextObject Object (page 224). The information passed will be used during formatting.

Syntax

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LineSpacing</td>
<td>Specifies the line spacing.</td>
</tr>
<tr>
<td>LineSpacingType</td>
<td>CRLineSpacingType (page 241). Specifies the line spacing type.</td>
</tr>
</tbody>
</table>

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

```vba
Sub SetText ( pText As String )
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pText</td>
<td>Specifies the text string for the Object. See Remarks below.</td>
</tr>
</tbody>
</table>

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**

The following enumerated types are discussed in this section:

- CRAlignment, Page 229
- CRAreaKind, Page 230
- CRBindingMatchType, Page 230
- CRCurrencySymbolType, Page 231
- CRDateOrder, Page 232
- CRDayType, Page 232
- CRDivisionMethod, Page 233
- CRExportDestinationType, Page 233
- CRFieldValueType, Page 235
- CRFieldKind, Page 235
- CRFieldMappingType, Page 235
- CRFieldName, Page 236
- CRGraphDataType, Page 237
- CRGraphType, Page 237
- CRLegendPosition, Page 241
- CRLinkJoinType, Page 241
- CRLinkLookUpType, Page 242
- CRMarkerShape, Page 242
- CRMinuteType, Page 242
- CRNegativeType, Page 243
- CRObjectKind, Page 244
- CRPaperOrientation, Page 244
- CRPaperSize, Page 244
- CRPaperSource, Page 246
CRParameterFieldType, Page 246  CRParameterPickListSortMethod, Page 246
CRPieLegendLayout, Page 247  CRPieSize, Page 247
CRPlaceHolderType, Page 247  CRPrinterDuplexType, Page 247
CRPrintingProgress, Page 247  CRRangeInfo, Page 248
CRReportResultType, Page 248  CRReportFileFormat, Page 248
CRReportKind, Page 248  CRReportVariableValueType, Page 249
CRRotationAngle, Page 249  CRRoundingType, Page 249
CRRunningTotalCondition, Page 250  CRSearchDirection, Page 250
CRSpecSecondType, Page 250  CRSliceDetachment, Page 250
CRSortDirection, Page 251  CRSpecialVarType, Page 251
CRSummaryType, Page 251  CRTableDifferences, Page 252
CRTextFormat, Page 253  CRTimeBase, Page 253
CRTopOrBottomNGroupSortOrder, Page 253  CRValueFormatType, Page 254
CRViewingAngle, Page 254  CRYearType, Page 254

**CRAlignment**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crDefaultAlign</td>
<td>0</td>
</tr>
<tr>
<td>crHorCenterAlign</td>
<td>2</td>
</tr>
<tr>
<td>crJustified</td>
<td>4</td>
</tr>
<tr>
<td>crLeftAlign</td>
<td>1</td>
</tr>
<tr>
<td>crRightAlign</td>
<td>3</td>
</tr>
</tbody>
</table>

**CRAMPMType**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crAmPmAfter</td>
<td>1</td>
</tr>
<tr>
<td>crAmPmBefore</td>
<td>0</td>
</tr>
</tbody>
</table>
**CRAreaKind**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crDetail</td>
<td>4</td>
</tr>
<tr>
<td>crGroupFooter</td>
<td>5</td>
</tr>
<tr>
<td>crGroupHeader</td>
<td>3</td>
</tr>
<tr>
<td>crPageFooter</td>
<td>7</td>
</tr>
<tr>
<td>crPageHeader</td>
<td>2</td>
</tr>
<tr>
<td>crReportFooter</td>
<td>8</td>
</tr>
<tr>
<td>crReportHeader</td>
<td>1</td>
</tr>
</tbody>
</table>

**CRBarSize**

<table>
<thead>
<tr>
<th>Constant (numeric order)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>crMinimumBarSize</td>
<td>0</td>
</tr>
<tr>
<td>crSmallBarSize</td>
<td>1</td>
</tr>
<tr>
<td>crAverageBarSize</td>
<td>2</td>
</tr>
<tr>
<td>crLargeBarSize</td>
<td>3</td>
</tr>
<tr>
<td>crMaximumBarSize</td>
<td>4</td>
</tr>
</tbody>
</table>

**CRBindingMatchType**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crBMTName</td>
<td>0</td>
</tr>
<tr>
<td>crBMTNameAndValue</td>
<td>1</td>
</tr>
</tbody>
</table>

**CRBooleanOutputType**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crOneOrZero</td>
<td>4</td>
</tr>
<tr>
<td>crTOrF</td>
<td>1</td>
</tr>
<tr>
<td>crTrueOrFalse</td>
<td>0</td>
</tr>
<tr>
<td>crYesOrNo</td>
<td>2</td>
</tr>
<tr>
<td>crYOrN</td>
<td>3</td>
</tr>
</tbody>
</table>
### CRConvertDateTimeType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crConvertDateTimeToDate</td>
<td>1</td>
</tr>
<tr>
<td>crConvertDateTimeToString</td>
<td>0</td>
</tr>
<tr>
<td>crKeepDateTimeType</td>
<td>2</td>
</tr>
</tbody>
</table>

### CRCurrencyPositionType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crLeadingCurrencyInsideNegative</td>
<td>0</td>
</tr>
<tr>
<td>crLeadingCurrencyOutsideNegative</td>
<td>1</td>
</tr>
<tr>
<td>crTrailingCurrencyInsideNegative</td>
<td>2</td>
</tr>
<tr>
<td>crTrailingCurrencyOutsideNegative</td>
<td>3</td>
</tr>
</tbody>
</table>

### CRCurrencySymbolType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crCSTFixedSymbol</td>
<td>1</td>
</tr>
<tr>
<td>crCSTFloatingSymbol</td>
<td>2</td>
</tr>
<tr>
<td>crCSTNoSymbol</td>
<td>0</td>
</tr>
</tbody>
</table>

### CRDatabaseType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crSQLDatabase</td>
<td>2</td>
</tr>
<tr>
<td>crStandardDatabase</td>
<td>1</td>
</tr>
</tbody>
</table>

### CRDateCalendarType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crGregorianCalendar</td>
<td>1</td>
</tr>
<tr>
<td>crGregorianUSCalendar</td>
<td>2</td>
</tr>
<tr>
<td>crHijriCalendar</td>
<td>6</td>
</tr>
<tr>
<td>crJapaneseCalendar</td>
<td>3</td>
</tr>
</tbody>
</table>
### CRDateEraType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crKoreanCalendar</td>
<td>5</td>
</tr>
<tr>
<td>crTaiwaneseCalendar</td>
<td>4</td>
</tr>
<tr>
<td>crThaiCalendar</td>
<td>7</td>
</tr>
</tbody>
</table>

### CRDateOrder

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crDayMonthYear</td>
<td>1</td>
</tr>
<tr>
<td>crMonthDayYear</td>
<td>2</td>
</tr>
<tr>
<td>crYearMonthDay</td>
<td>0</td>
</tr>
</tbody>
</table>

### CRDateWindowsDefaultType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crNotUsingWindowsDefaults</td>
<td>2</td>
</tr>
<tr>
<td>crUseWindowsLongDate</td>
<td>0</td>
</tr>
<tr>
<td>crUseWindowsShortDate</td>
<td>1</td>
</tr>
</tbody>
</table>

### CRDayType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crLeadingZeroNumericDay</td>
<td>1</td>
</tr>
<tr>
<td>crNoDay</td>
<td>2</td>
</tr>
<tr>
<td>crNumericDay</td>
<td>0</td>
</tr>
</tbody>
</table>
**CRDiscreteOrRangeKind**

<table>
<thead>
<tr>
<th>Constant (numeric order)</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crDiscreteValue</td>
<td>0</td>
</tr>
<tr>
<td>crRangeValue</td>
<td>1</td>
</tr>
<tr>
<td>crDiscreteAndRangeValue</td>
<td>2</td>
</tr>
</tbody>
</table>

**CRDivisionMethod**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crAutomaticDivision</td>
<td>0</td>
</tr>
<tr>
<td>crManualDivision</td>
<td>1</td>
</tr>
</tbody>
</table>

**CRExchangeDestinationType**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crExchangeFolderType</td>
<td>0</td>
</tr>
<tr>
<td>crExchangePostDocMessage</td>
<td>1011</td>
</tr>
</tbody>
</table>

**CRExportDestinationType**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crEDTApplication</td>
<td>5</td>
</tr>
<tr>
<td>crEDTDiskFile</td>
<td>1</td>
</tr>
<tr>
<td>crEDTEmailMAPI</td>
<td>2</td>
</tr>
<tr>
<td>crEDTEmailVIM</td>
<td>3</td>
</tr>
<tr>
<td>crEDTLotusNotes</td>
<td>6</td>
</tr>
<tr>
<td>crEDTMicrosoftExchange</td>
<td>4</td>
</tr>
<tr>
<td>crEDTNoDestination</td>
<td>0</td>
</tr>
</tbody>
</table>
## CRExportFormatType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crEFTCharSeparatedValues</td>
<td>7</td>
</tr>
<tr>
<td>crEFTCommaSeparatedValues</td>
<td>5</td>
</tr>
<tr>
<td>crEFTCrystalReport</td>
<td>1</td>
</tr>
<tr>
<td>crEFTDataInterchange</td>
<td>2</td>
</tr>
<tr>
<td>crEFTExcel21</td>
<td>18</td>
</tr>
<tr>
<td>crEFTExcel30</td>
<td>19</td>
</tr>
<tr>
<td>crEFTExcel40</td>
<td>20</td>
</tr>
<tr>
<td>crEFTExcel50</td>
<td>21</td>
</tr>
<tr>
<td>crEFTExcel50Tabular</td>
<td>22</td>
</tr>
<tr>
<td>crEFTExcel70</td>
<td>27</td>
</tr>
<tr>
<td>crEFTExcel70Tabular</td>
<td>28</td>
</tr>
<tr>
<td>crEFTExcel80</td>
<td>29</td>
</tr>
<tr>
<td>crEFTExcel80Tabular</td>
<td>30</td>
</tr>
<tr>
<td>crEFTExplorer32Extend</td>
<td>25</td>
</tr>
<tr>
<td>crEFTHTML32Standard</td>
<td>24</td>
</tr>
<tr>
<td>crEFTLotus123WK1</td>
<td>12</td>
</tr>
<tr>
<td>crEFTLotus123WK3</td>
<td>13</td>
</tr>
<tr>
<td>crEFTLotus123WKS</td>
<td>11</td>
</tr>
<tr>
<td>crEFTNetScape20</td>
<td>26</td>
</tr>
<tr>
<td>crEFTNoFormat</td>
<td>0</td>
</tr>
<tr>
<td>crEFTODBC</td>
<td>23</td>
</tr>
<tr>
<td>crEFTPaginatedText</td>
<td>10</td>
</tr>
<tr>
<td>crEFTRecordStyle</td>
<td>3</td>
</tr>
<tr>
<td>crEFTRichText</td>
<td>4</td>
</tr>
<tr>
<td>crEFTTabSeparatedText</td>
<td>9</td>
</tr>
<tr>
<td>crEFTTabSeparatedValues</td>
<td>6</td>
</tr>
<tr>
<td>crEFTText</td>
<td>8</td>
</tr>
<tr>
<td>crEFTWordForWindows</td>
<td>14</td>
</tr>
</tbody>
</table>
**CRFieldKind**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crDatabaseField</td>
<td>1</td>
</tr>
<tr>
<td>crFormulaField</td>
<td>2</td>
</tr>
<tr>
<td>crGroupNameField</td>
<td>5</td>
</tr>
<tr>
<td>crRunningTotalField</td>
<td>7</td>
</tr>
<tr>
<td>crParameterField</td>
<td>6</td>
</tr>
<tr>
<td>crSpecialVarField</td>
<td>4</td>
</tr>
<tr>
<td>crSQLExpressionField</td>
<td>8</td>
</tr>
<tr>
<td>crSummaryField</td>
<td>3</td>
</tr>
</tbody>
</table>

**CRFieldMappingType**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crAutoFieldMapping</td>
<td>0</td>
</tr>
<tr>
<td>crEventFieldMapping</td>
<td>2</td>
</tr>
<tr>
<td>crPromptFieldMapping</td>
<td>1</td>
</tr>
</tbody>
</table>

**CRFieldValueType**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crBitmapField</td>
<td>17</td>
</tr>
<tr>
<td>crBlobField</td>
<td>15</td>
</tr>
<tr>
<td>crBooleanField</td>
<td>9</td>
</tr>
<tr>
<td>crChartField</td>
<td>21</td>
</tr>
<tr>
<td>crCurrencyField</td>
<td>8</td>
</tr>
<tr>
<td>crDateTimeField</td>
<td>10</td>
</tr>
<tr>
<td>crDateTimeField</td>
<td>16</td>
</tr>
<tr>
<td>crIconField</td>
<td>18</td>
</tr>
<tr>
<td>Constant</td>
<td>Value</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------</td>
</tr>
<tr>
<td>crInt16sField</td>
<td>3</td>
</tr>
<tr>
<td>crInt16uField</td>
<td>4</td>
</tr>
<tr>
<td>crInt32sField</td>
<td>5</td>
</tr>
<tr>
<td>crInt32uField</td>
<td>6</td>
</tr>
<tr>
<td>crInt8sField</td>
<td>1</td>
</tr>
<tr>
<td>crInt8uField</td>
<td>2</td>
</tr>
<tr>
<td>crNumberField</td>
<td>7</td>
</tr>
<tr>
<td>crOleField</td>
<td>20</td>
</tr>
<tr>
<td>crPersistentMemoField</td>
<td>14</td>
</tr>
<tr>
<td>crPictureField</td>
<td>19</td>
</tr>
<tr>
<td>crStringField</td>
<td>12</td>
</tr>
<tr>
<td>crTimeField</td>
<td>11</td>
</tr>
<tr>
<td>crTransientMemoField</td>
<td>13</td>
</tr>
<tr>
<td>crUnknownField</td>
<td>22</td>
</tr>
</tbody>
</table>

**CRFormulaSyntax**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crBasicSyntaxFormula</td>
<td>1</td>
</tr>
<tr>
<td>crCrystalSyntaxFormula</td>
<td>0</td>
</tr>
</tbody>
</table>

**CRGraphColor**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crBlackAndWhiteGraph</td>
<td>1</td>
</tr>
<tr>
<td>crColorGraph</td>
<td>0</td>
</tr>
</tbody>
</table>

**CRGraphDataPoint**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crNone</td>
<td>0</td>
</tr>
<tr>
<td>crShowLabel</td>
<td>1</td>
</tr>
<tr>
<td>crShowValue</td>
<td>2</td>
</tr>
</tbody>
</table>
### CRGraphDataType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crCrossTabGraph</td>
<td>2</td>
</tr>
<tr>
<td>crDetailGraph</td>
<td>1</td>
</tr>
<tr>
<td>crGroupGraph</td>
<td>0</td>
</tr>
</tbody>
</table>

### CRGraphDirection

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crHorizontalGraph</td>
<td>0</td>
</tr>
<tr>
<td>crVerticalGraph</td>
<td>1</td>
</tr>
</tbody>
</table>

### CRGraphType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crAbsoluteAreaGraph</td>
<td>20</td>
</tr>
<tr>
<td>crDualAxisBubbleGraph</td>
<td>91</td>
</tr>
<tr>
<td>crFaked3DAbsoluteAreaGraph</td>
<td>23</td>
</tr>
<tr>
<td>crFaked3DPercentAreaGraph</td>
<td>25</td>
</tr>
<tr>
<td>crFaked3DPercentBarGraph</td>
<td>5</td>
</tr>
<tr>
<td>crFaked3DRegularPieGraph</td>
<td>31</td>
</tr>
<tr>
<td>crFaked3DSideBySideBarGraph</td>
<td>3</td>
</tr>
<tr>
<td>crFaked3DStackedAreaGraph</td>
<td>24</td>
</tr>
<tr>
<td>crFaked3DStackedBarGraph</td>
<td>4</td>
</tr>
<tr>
<td>crHighLowDualAxisGraph</td>
<td>101</td>
</tr>
<tr>
<td>Obsolete.</td>
<td></td>
</tr>
<tr>
<td>crHighLowGraph</td>
<td>100</td>
</tr>
<tr>
<td>crHighLowOpenCloseDualAxisGraph</td>
<td>105</td>
</tr>
<tr>
<td>Obsolete.</td>
<td></td>
</tr>
<tr>
<td>crHighLowOpenCloseGraph</td>
<td>104</td>
</tr>
<tr>
<td>crHighLowOpenDualAxisGraph</td>
<td>103</td>
</tr>
<tr>
<td>Obsolete.</td>
<td></td>
</tr>
<tr>
<td>crHighLowOpenGraph</td>
<td>102</td>
</tr>
<tr>
<td>Obsolete.</td>
<td></td>
</tr>
<tr>
<td>crLineGraphWithMarkers</td>
<td>13</td>
</tr>
<tr>
<td>crMultipleDoughnutGraph</td>
<td>41</td>
</tr>
<tr>
<td>crMultiplePieGraph</td>
<td>32</td>
</tr>
<tr>
<td>crMultipleProportionalDoughnutGraph</td>
<td>42</td>
</tr>
</tbody>
</table>
### RDC Automation Server

#### CRGridlineType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crMajorAndMinorGridlines</td>
<td>3</td>
</tr>
<tr>
<td>crMajorGridlines</td>
<td>2</td>
</tr>
</tbody>
</table>

### CRGridlineType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crMultipleProportionalPieGraph</td>
<td>33</td>
</tr>
<tr>
<td>crPercentageLineGraph</td>
<td>12</td>
</tr>
<tr>
<td>crPercentageLineGraphWithMarkers</td>
<td>15</td>
</tr>
<tr>
<td>crPercentAreaGraph</td>
<td>22</td>
</tr>
<tr>
<td>crPercentBarGraph</td>
<td>2</td>
</tr>
<tr>
<td>crRadarDualAxisGraph</td>
<td>82</td>
</tr>
<tr>
<td>crRegularBubbleGraph</td>
<td>90</td>
</tr>
<tr>
<td>crRegularDoughnutGraph</td>
<td>40</td>
</tr>
<tr>
<td>crRegularLineGraph</td>
<td>10</td>
</tr>
<tr>
<td>crRegularPieGraph</td>
<td>30</td>
</tr>
<tr>
<td>crRegularRadarGraph</td>
<td>80</td>
</tr>
<tr>
<td>crSideBySideBarGraph</td>
<td>0</td>
</tr>
<tr>
<td>crStackedAreaGraph</td>
<td>21</td>
</tr>
<tr>
<td>crStackedBarGraph</td>
<td>1</td>
</tr>
<tr>
<td>crStackedLineGraph</td>
<td>11</td>
</tr>
<tr>
<td>crStackedLineGraphWithMarkers</td>
<td>14</td>
</tr>
<tr>
<td>crStackedRadarGraph</td>
<td>81</td>
</tr>
<tr>
<td>crThreeDCutCornersGraph</td>
<td>53</td>
</tr>
<tr>
<td>crThreeDOctagonGraph</td>
<td>52</td>
</tr>
<tr>
<td>crThreeDPyramidGraph</td>
<td>51</td>
</tr>
<tr>
<td>crThreeDRegularGraph</td>
<td>50</td>
</tr>
<tr>
<td>crThreeDSurfaceHoneycombGraph</td>
<td>62</td>
</tr>
<tr>
<td>crThreeDSurfaceRegularGraph</td>
<td>60</td>
</tr>
<tr>
<td>crThreeDSurfaceWithSidesGraph</td>
<td>61</td>
</tr>
<tr>
<td>crUnknownGraph</td>
<td>1000</td>
</tr>
<tr>
<td>crXyScatterDualAxisGraph</td>
<td>71</td>
</tr>
<tr>
<td>crXyScatterDualAxisWithLabelsGraph</td>
<td>73</td>
</tr>
<tr>
<td>crXyScatterGraph</td>
<td>70</td>
</tr>
<tr>
<td>crXyScatterWithLabelsGraph</td>
<td>72</td>
</tr>
<tr>
<td>Constant</td>
<td>Value</td>
</tr>
<tr>
<td>----------------</td>
<td>-------</td>
</tr>
<tr>
<td>crMinorGridlines</td>
<td>1</td>
</tr>
<tr>
<td>crNoGridlines</td>
<td>0</td>
</tr>
</tbody>
</table>

**CRGroupCondition**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crGCAnnually</td>
<td>7</td>
</tr>
<tr>
<td>crGCAnyValue</td>
<td>14</td>
</tr>
<tr>
<td>crGCBiweekly</td>
<td>2</td>
</tr>
<tr>
<td>crGCBYAMPM</td>
<td>18</td>
</tr>
<tr>
<td>crGCBYHour</td>
<td>17</td>
</tr>
<tr>
<td>crGCBYMinute</td>
<td>16</td>
</tr>
<tr>
<td>crGCBYSecond</td>
<td>15</td>
</tr>
<tr>
<td>crGCDaily</td>
<td>0</td>
</tr>
<tr>
<td>crGCEveryNo</td>
<td>11</td>
</tr>
<tr>
<td>crGCEveryYes</td>
<td>10</td>
</tr>
<tr>
<td>crGCMonthly</td>
<td>4</td>
</tr>
<tr>
<td>crGCNextIsNo</td>
<td>13</td>
</tr>
<tr>
<td>crGCNextIsYes</td>
<td>12</td>
</tr>
<tr>
<td>crGCQuarterly</td>
<td>5</td>
</tr>
<tr>
<td>crGCSemiAnnually</td>
<td>6</td>
</tr>
<tr>
<td>crGCSemimonthly</td>
<td>3</td>
</tr>
<tr>
<td>crGCToNo</td>
<td>9</td>
</tr>
<tr>
<td>crGCToNo</td>
<td>8</td>
</tr>
<tr>
<td>crGCWeekly</td>
<td>1</td>
</tr>
</tbody>
</table>

**CRHourType**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Const crNoHour</td>
<td>2</td>
</tr>
<tr>
<td>Const cr_numericHour</td>
<td>0</td>
</tr>
<tr>
<td>Const cr_numericHourNoLeadingZero</td>
<td>1</td>
</tr>
</tbody>
</table>
### CRHTMLPageStyle

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crFramePageStyle</td>
<td>2</td>
</tr>
<tr>
<td>crPlainPageStyle</td>
<td>0</td>
</tr>
<tr>
<td>crToolbarAtBottomPageStyle</td>
<td>4</td>
</tr>
<tr>
<td>crToolbarAtTopPageStyle</td>
<td>3</td>
</tr>
<tr>
<td>crToolbarPageStyle</td>
<td>1</td>
</tr>
</tbody>
</table>

### CRHTMLToolbarStyle

These bitwise constants can be XOR’d to specify the toolbar style.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crToolbarRefreshButton</td>
<td>1</td>
</tr>
<tr>
<td>crToolbarSearchBox</td>
<td>2</td>
</tr>
</tbody>
</table>

### CRImageType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crDIBImageType</td>
<td>1</td>
</tr>
<tr>
<td>crJPEGImageType</td>
<td>2</td>
</tr>
</tbody>
</table>

### CRLeadingDayPosition

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crLeadingDayOfWeek</td>
<td>0</td>
</tr>
<tr>
<td>crTrailingDayOfWeek</td>
<td>1</td>
</tr>
</tbody>
</table>

### CRLeadingDayType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crLongLeadingDay</td>
<td>1</td>
</tr>
<tr>
<td>crNoLeadingDay</td>
<td>2</td>
</tr>
<tr>
<td>crShortLeadingDay</td>
<td>0</td>
</tr>
</tbody>
</table>
### CRLegendPosition

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crPlaceBottomCenter</td>
<td>1</td>
</tr>
<tr>
<td>crPlaceLeft</td>
<td>4</td>
</tr>
<tr>
<td>crPlaceRight</td>
<td>3</td>
</tr>
<tr>
<td>crPlaceTopCenter</td>
<td>2</td>
</tr>
<tr>
<td>crPlaceUpperRight</td>
<td>0</td>
</tr>
</tbody>
</table>

### CRLineSpacingType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crExactSpacing</td>
<td>1</td>
</tr>
<tr>
<td>crMultipleSpacing</td>
<td>0</td>
</tr>
</tbody>
</table>

### CRLineStyle

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crLSDashLine</td>
<td>3</td>
</tr>
<tr>
<td>crLSDotLine</td>
<td>4</td>
</tr>
<tr>
<td>crLSNoLine</td>
<td>0. Not valid for LineObject.LineStyle.</td>
</tr>
<tr>
<td>crLSSingleLine</td>
<td>1</td>
</tr>
</tbody>
</table>

### CRLinkJoinType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crJROuter</td>
<td>7</td>
</tr>
<tr>
<td>crJTEqual</td>
<td>4</td>
</tr>
<tr>
<td>crJTGreaterOrEqual</td>
<td>10</td>
</tr>
<tr>
<td>crJTGreaterThan</td>
<td>8</td>
</tr>
<tr>
<td>crJTLessOuter</td>
<td>5</td>
</tr>
<tr>
<td>crJTLessOrEqual</td>
<td>11</td>
</tr>
<tr>
<td>crJTLessThan</td>
<td>9</td>
</tr>
<tr>
<td>crJTNotEqual</td>
<td>12</td>
</tr>
<tr>
<td>crJTRightOuter</td>
<td>6</td>
</tr>
<tr>
<td>Constant</td>
<td>Value</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>crLTLookupParallel</td>
<td>1</td>
</tr>
<tr>
<td>crLTLookupProduct</td>
<td>2</td>
</tr>
<tr>
<td>crLTLookupSeries</td>
<td>3</td>
</tr>
</tbody>
</table>

### CRMarkerShape

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crCircleShape</td>
<td>4</td>
</tr>
<tr>
<td>crDiamondShape</td>
<td>5</td>
</tr>
<tr>
<td>crRectangleShape</td>
<td>1</td>
</tr>
<tr>
<td>crTriangleShape</td>
<td>8</td>
</tr>
</tbody>
</table>

### CRMarkerSize

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crLargeMarkers</td>
<td>4</td>
</tr>
<tr>
<td>crMediumLargeMarkers</td>
<td>3</td>
</tr>
<tr>
<td>crMediumMarkers</td>
<td>2</td>
</tr>
<tr>
<td>crMediumSmallMarkers</td>
<td>1</td>
</tr>
<tr>
<td>crSmallMarkers</td>
<td>0</td>
</tr>
</tbody>
</table>

### CRMinuteType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crNoMinute</td>
<td>2</td>
</tr>
<tr>
<td>crNumericMinute</td>
<td>0</td>
</tr>
<tr>
<td>crNumericMinuteNoLeadingZero</td>
<td>1</td>
</tr>
</tbody>
</table>
**CRMonthType**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crLeadingZeroNumericMonth</td>
<td>1</td>
</tr>
<tr>
<td>crLongMonth</td>
<td>3</td>
</tr>
<tr>
<td>crNoMonth</td>
<td>4</td>
</tr>
<tr>
<td>crNumericMonth</td>
<td>0</td>
</tr>
<tr>
<td>crShortMonth</td>
<td>2</td>
</tr>
</tbody>
</table>

**CRNegativeType**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crBracketed</td>
<td>3</td>
</tr>
<tr>
<td>crLeadingMinus</td>
<td>1</td>
</tr>
<tr>
<td>crNotNegative</td>
<td>0</td>
</tr>
<tr>
<td>crTrailingMinus</td>
<td>2</td>
</tr>
</tbody>
</table>

**CRNumberFormat**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crCurrencyMillions</td>
<td>12</td>
</tr>
<tr>
<td>crCurrencyNoDecimal</td>
<td>3</td>
</tr>
<tr>
<td>crCurrencyThousands</td>
<td>11</td>
</tr>
<tr>
<td>crCurrencyTwoDecimal</td>
<td>4</td>
</tr>
<tr>
<td>crCustomNumberFormat</td>
<td>8</td>
</tr>
<tr>
<td>crMillionsNoDecimal</td>
<td>10</td>
</tr>
<tr>
<td>crNoDecimal</td>
<td>0</td>
</tr>
<tr>
<td>crOneDecimal</td>
<td>1</td>
</tr>
<tr>
<td>crPercentNoDecimal</td>
<td>5</td>
</tr>
<tr>
<td>crPercentOneDecimal</td>
<td>6</td>
</tr>
<tr>
<td>crPercentTwoDecimal</td>
<td>7</td>
</tr>
<tr>
<td>crThousandsNoDecimal</td>
<td>9</td>
</tr>
<tr>
<td>crTwoDecimal</td>
<td>2</td>
</tr>
</tbody>
</table>
### CRObjektKind

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crBlobFieldObject</td>
<td>9</td>
</tr>
<tr>
<td>crBoxObject</td>
<td>4</td>
</tr>
<tr>
<td>crCrossTabObject</td>
<td>8</td>
</tr>
<tr>
<td>crFieldObject</td>
<td>1</td>
</tr>
<tr>
<td>crGraphObject</td>
<td>7</td>
</tr>
<tr>
<td>crLineObject</td>
<td>3</td>
</tr>
<tr>
<td>crMapObject</td>
<td>10</td>
</tr>
<tr>
<td>crOlapGridObject</td>
<td>11</td>
</tr>
<tr>
<td>crOleObject</td>
<td>6</td>
</tr>
<tr>
<td>crSubreportObject</td>
<td>5</td>
</tr>
<tr>
<td>crTextObject</td>
<td>2</td>
</tr>
</tbody>
</table>

### CRPaperOrientation

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crDefaultPaperOrientation</td>
<td>0</td>
</tr>
<tr>
<td>crLandscape</td>
<td>2</td>
</tr>
<tr>
<td>crPortrait</td>
<td>1</td>
</tr>
</tbody>
</table>

### CRPaperSize

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crDefaultPaperSize</td>
<td>0</td>
</tr>
<tr>
<td>crPaper10x14</td>
<td>16</td>
</tr>
<tr>
<td>crPaper11x17</td>
<td>17</td>
</tr>
<tr>
<td>crPaperA3</td>
<td>8</td>
</tr>
<tr>
<td>crPaperA4</td>
<td>9</td>
</tr>
<tr>
<td>crPaperA4Small</td>
<td>10</td>
</tr>
<tr>
<td>crPaperA5</td>
<td>11</td>
</tr>
<tr>
<td>crPaperB4</td>
<td>12</td>
</tr>
<tr>
<td>crPaperB5</td>
<td>13</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----</td>
</tr>
<tr>
<td>crPaperCsheet</td>
<td>24</td>
</tr>
<tr>
<td>crPaperDsheet</td>
<td>25</td>
</tr>
<tr>
<td>crPaperEnvelope10</td>
<td>20</td>
</tr>
<tr>
<td>crPaperEnvelope11</td>
<td>21</td>
</tr>
<tr>
<td>crPaperEnvelope12</td>
<td>22</td>
</tr>
<tr>
<td>crPaperEnvelope14</td>
<td>23</td>
</tr>
<tr>
<td>crPaperEnvelope9</td>
<td>19</td>
</tr>
<tr>
<td>crPaperEnvelopeB4</td>
<td>33</td>
</tr>
<tr>
<td>crPaperEnvelopeB5</td>
<td>34</td>
</tr>
<tr>
<td>crPaperEnvelopeB6</td>
<td>35</td>
</tr>
<tr>
<td>crPaperEnvelopeC3</td>
<td>29</td>
</tr>
<tr>
<td>crPaperEnvelopeC4</td>
<td>30</td>
</tr>
<tr>
<td>crPaperEnvelopeC5</td>
<td>28</td>
</tr>
<tr>
<td>crPaperEnvelopeC6</td>
<td>31</td>
</tr>
<tr>
<td>crPaperEnvelopeC65</td>
<td>32</td>
</tr>
<tr>
<td>crPaperEnvelopeDL</td>
<td>27</td>
</tr>
<tr>
<td>crPaperEnvelopeItaly</td>
<td>36</td>
</tr>
<tr>
<td>crPaperEnvelopeMonarch</td>
<td>37</td>
</tr>
<tr>
<td>crPaperEnvelopePersonal</td>
<td>38</td>
</tr>
<tr>
<td>crPaperEsheet</td>
<td>26</td>
</tr>
<tr>
<td>crPaperExecutive</td>
<td>7</td>
</tr>
<tr>
<td>crPaperFanfoldLegalGerman</td>
<td>41</td>
</tr>
<tr>
<td>crPaperFanfoldStdGerman</td>
<td>40</td>
</tr>
<tr>
<td>crPaperFanfoldUS</td>
<td>39</td>
</tr>
<tr>
<td>crPaperFolio</td>
<td>14</td>
</tr>
<tr>
<td>crPaperLedger</td>
<td>4</td>
</tr>
<tr>
<td>crPaperLegal</td>
<td>5</td>
</tr>
<tr>
<td>crPaperLetter</td>
<td>1</td>
</tr>
<tr>
<td>crPaperLetterSmall</td>
<td>2</td>
</tr>
<tr>
<td>crPaperNote</td>
<td>18</td>
</tr>
<tr>
<td>crPaperQuarto</td>
<td>15</td>
</tr>
<tr>
<td>crPaperStatement</td>
<td>6</td>
</tr>
<tr>
<td>crPaperTabloid</td>
<td>3</td>
</tr>
</tbody>
</table>
**CRPaperSource**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crPRBinAuto</td>
<td>7</td>
</tr>
<tr>
<td>crPRBinCassette</td>
<td>14</td>
</tr>
<tr>
<td>crPRBinEnvelope</td>
<td>5</td>
</tr>
<tr>
<td>crPRBinEnvManual</td>
<td>6</td>
</tr>
<tr>
<td>crPRBinFormSource</td>
<td>15</td>
</tr>
<tr>
<td>crPRBinLargeCapacity</td>
<td>11</td>
</tr>
<tr>
<td>crPRBinLargeFmt</td>
<td>10</td>
</tr>
<tr>
<td>crPRBinLower</td>
<td>2</td>
</tr>
<tr>
<td>crPRBinManual</td>
<td>4</td>
</tr>
<tr>
<td>crPRBinMiddle</td>
<td>3</td>
</tr>
<tr>
<td>crPRBinSmallFmt</td>
<td>9</td>
</tr>
<tr>
<td>crPRBinTractor</td>
<td>8</td>
</tr>
<tr>
<td>crPRBinUpper</td>
<td>1</td>
</tr>
</tbody>
</table>

**CRParameterFieldType**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crQueryParameter</td>
<td>1</td>
</tr>
<tr>
<td>crReportParameter</td>
<td>0</td>
</tr>
<tr>
<td>crStoreProcedureParameter</td>
<td>2</td>
</tr>
</tbody>
</table>

**CRParameterPickListSortMethod**

<table>
<thead>
<tr>
<th>Constant (numeric order)</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crNoSort</td>
<td>0</td>
</tr>
<tr>
<td>crAlphanumericAscending</td>
<td>1</td>
</tr>
<tr>
<td>crAlphanumericDescending</td>
<td>2</td>
</tr>
<tr>
<td>crNumericAscending</td>
<td>3</td>
</tr>
<tr>
<td>crNumericDescending</td>
<td>4</td>
</tr>
</tbody>
</table>
### CRPieLegendLayout

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crAmountLayout</td>
<td>1</td>
</tr>
<tr>
<td>crCustomLayout</td>
<td>2</td>
</tr>
<tr>
<td>crPercentLayout</td>
<td>0</td>
</tr>
</tbody>
</table>

### CRPieSize

<table>
<thead>
<tr>
<th>Constant (numeric order)</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crMaximumPieSize</td>
<td>0</td>
</tr>
<tr>
<td>crLargePieSize</td>
<td>16</td>
</tr>
<tr>
<td>crAveragePieSize</td>
<td>32</td>
</tr>
<tr>
<td>crSmallPieSize</td>
<td>48</td>
</tr>
<tr>
<td>crMinimumPieSize</td>
<td>64</td>
</tr>
</tbody>
</table>

### CRPlaceHolderType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crAllowPlaceHolders</td>
<td>2</td>
</tr>
<tr>
<td>crDelayTotalPageCountCalc</td>
<td>1</td>
</tr>
</tbody>
</table>

### CRPrinterDuplexType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crPRDPDefault</td>
<td>0</td>
</tr>
<tr>
<td>crPRDPHorizontal</td>
<td>3</td>
</tr>
<tr>
<td>crPRDPSimplex</td>
<td>1</td>
</tr>
<tr>
<td>crPRDPVertical</td>
<td>2</td>
</tr>
</tbody>
</table>

### CRPrintingProgress

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crPrintingCancelled</td>
<td>5</td>
</tr>
<tr>
<td>crPrintingCompleted</td>
<td>3</td>
</tr>
</tbody>
</table>
### CRRangeInfo

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crRangeNotIncludeUpperLowerBound</td>
<td>0</td>
</tr>
<tr>
<td>crRangeIncludeUpperBound</td>
<td>1</td>
</tr>
<tr>
<td>crRangeIncludeLowerBound</td>
<td>2</td>
</tr>
<tr>
<td>crRangeNoUpperBound</td>
<td>4</td>
</tr>
<tr>
<td>crRangeNoLowerBound</td>
<td>8</td>
</tr>
</tbody>
</table>

### CRRenderResultType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crBSTRType</td>
<td>8</td>
</tr>
<tr>
<td>crUISafeArrayType</td>
<td>8209</td>
</tr>
</tbody>
</table>

### CRReportFileFormat

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>cr70FileFormat</td>
<td>1792</td>
</tr>
<tr>
<td>cr80FileFormat</td>
<td>2048</td>
</tr>
</tbody>
</table>

### CRReportKind

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crColumnarReport</td>
<td>1</td>
</tr>
<tr>
<td>crLabelReport</td>
<td>2</td>
</tr>
<tr>
<td>crMulColumnReport</td>
<td>3</td>
</tr>
</tbody>
</table>
CRRReportVariableValueType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crRVBoolean</td>
<td>2</td>
</tr>
<tr>
<td>crRVCurrency</td>
<td>1</td>
</tr>
<tr>
<td>crRVDate</td>
<td>3</td>
</tr>
<tr>
<td>crRVDateTime</td>
<td>5</td>
</tr>
<tr>
<td>crRVNumber</td>
<td>0</td>
</tr>
<tr>
<td>crRVString</td>
<td>6</td>
</tr>
<tr>
<td>crRVTime</td>
<td>4</td>
</tr>
</tbody>
</table>

CRRotationAngle

<table>
<thead>
<tr>
<th>Constant (numeric order)</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crRotate0</td>
<td>0</td>
</tr>
<tr>
<td>crRotate90</td>
<td>1</td>
</tr>
<tr>
<td>crRotate270</td>
<td>2</td>
</tr>
</tbody>
</table>

CRRoundingType

<table>
<thead>
<tr>
<th>Constant (numeric order)</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crRoundToMillion</td>
<td>17</td>
</tr>
<tr>
<td>crRoundToHundredThousand</td>
<td>16</td>
</tr>
<tr>
<td>crRoundToTenThousand</td>
<td>15</td>
</tr>
<tr>
<td>crRoundToThousand</td>
<td>14</td>
</tr>
<tr>
<td>crRoundToHundred</td>
<td>13</td>
</tr>
<tr>
<td>crRoundToTen</td>
<td>12</td>
</tr>
<tr>
<td>crRoundToUnit</td>
<td>11</td>
</tr>
<tr>
<td>crRoundToTenth</td>
<td>10</td>
</tr>
<tr>
<td>crRoundToHundredth</td>
<td>9</td>
</tr>
<tr>
<td>crRoundToThousandth</td>
<td>8</td>
</tr>
<tr>
<td>crRoundToTenThousandth</td>
<td>7</td>
</tr>
<tr>
<td>crRoundToHundredThousand</td>
<td>6</td>
</tr>
<tr>
<td>crRoundToMillionth</td>
<td>5</td>
</tr>
<tr>
<td>crRoundToTenMillionth</td>
<td>4</td>
</tr>
<tr>
<td>Constant (numeric order)</td>
<td>Value</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>crRoundToHundredMillionth</td>
<td>3</td>
</tr>
<tr>
<td>crRoundTo Billionth</td>
<td>2</td>
</tr>
<tr>
<td>crRoundToTenBillionth</td>
<td>1</td>
</tr>
</tbody>
</table>

### CRRunningTotalCondition

<table>
<thead>
<tr>
<th>Constant (numeric order)</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crRTEvalNoCondition</td>
<td>0</td>
</tr>
<tr>
<td>crRTEvalOnChangeOfField</td>
<td>1</td>
</tr>
<tr>
<td>crRTEvalOnChangeOfGroup</td>
<td>2</td>
</tr>
<tr>
<td>crRTEvalOnFormula</td>
<td>3</td>
</tr>
</tbody>
</table>

### CRSearchDirection

<table>
<thead>
<tr>
<th>Constant (numeric order)</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crForward</td>
<td>0</td>
</tr>
<tr>
<td>crBackward</td>
<td>1</td>
</tr>
</tbody>
</table>

### CRSecondType

<table>
<thead>
<tr>
<th>Constant (numeric order)</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crNumericNoSecond</td>
<td>2</td>
</tr>
<tr>
<td>crNumericSecond</td>
<td>0</td>
</tr>
<tr>
<td>crNumericSecondNoLeadingZero</td>
<td>1</td>
</tr>
</tbody>
</table>

### CRSliceDetachment

<table>
<thead>
<tr>
<th>Constant (numeric order)</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crLargestSlice</td>
<td>2</td>
</tr>
<tr>
<td>crSmallestSlice</td>
<td>1</td>
</tr>
<tr>
<td>crNoDetachment</td>
<td>0</td>
</tr>
</tbody>
</table>
### CRSortDirection

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crAscendingOrder</td>
<td>0</td>
</tr>
<tr>
<td>crDescendingOrder</td>
<td>1</td>
</tr>
<tr>
<td>crOriginalOrder</td>
<td>2. Not supported for CrossTabGroups.</td>
</tr>
<tr>
<td>crSpecifiedOrder</td>
<td>3. Not supported for CrossTabGroups.</td>
</tr>
</tbody>
</table>

### CRSpecialVarType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crSVTDataDate</td>
<td>4</td>
</tr>
<tr>
<td>crSVTDataTime</td>
<td>5</td>
</tr>
<tr>
<td>crSVTFileAuthor</td>
<td>15</td>
</tr>
<tr>
<td>crSVTFileCreationDate</td>
<td>16 (&amp;H10)</td>
</tr>
<tr>
<td>crSVTFilename</td>
<td>14</td>
</tr>
<tr>
<td>crSVTGroupNumber</td>
<td>8</td>
</tr>
<tr>
<td>crSVTGroupSelection</td>
<td>13</td>
</tr>
<tr>
<td>crSVMODificationDate</td>
<td>2</td>
</tr>
<tr>
<td>crSVMODificationTime</td>
<td>3</td>
</tr>
<tr>
<td>crSVTPageNofM</td>
<td>17 (&amp;H11)</td>
</tr>
<tr>
<td>crSVTPageNumber</td>
<td>7</td>
</tr>
<tr>
<td>crSVPDDate</td>
<td>0</td>
</tr>
<tr>
<td>crSVPDTime</td>
<td>1</td>
</tr>
<tr>
<td>crSVTRecordNumber</td>
<td>6</td>
</tr>
<tr>
<td>crSVTRecordSelection</td>
<td>12</td>
</tr>
<tr>
<td>crSVTReportComments</td>
<td>11</td>
</tr>
<tr>
<td>crSVTReportTitle</td>
<td>10</td>
</tr>
<tr>
<td>crSVTTotalPageCount</td>
<td>9</td>
</tr>
</tbody>
</table>

### CRSummaryType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crSTAverage</td>
<td>1</td>
</tr>
<tr>
<td>crSTCount</td>
<td>6</td>
</tr>
</tbody>
</table>
## Constant Value

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crSTDCorrelation</td>
<td>10</td>
</tr>
<tr>
<td>crSTDCovariance</td>
<td>11</td>
</tr>
<tr>
<td>crSTDCount</td>
<td>9</td>
</tr>
<tr>
<td>crSTDMedian</td>
<td>13</td>
</tr>
<tr>
<td>crSTDMode</td>
<td>17</td>
</tr>
<tr>
<td>crSTDNthLargest</td>
<td>15</td>
</tr>
<tr>
<td>crSTDNthMostFrequent</td>
<td>18</td>
</tr>
<tr>
<td>crSTDNthSmallest</td>
<td>16</td>
</tr>
<tr>
<td>crSTDPercentage</td>
<td>19</td>
</tr>
<tr>
<td>crSTDPercenitile</td>
<td>14</td>
</tr>
<tr>
<td>crSTDWeightedAvg</td>
<td>12</td>
</tr>
<tr>
<td>crSTMaximum</td>
<td>4</td>
</tr>
<tr>
<td>crSTMinimum</td>
<td>5</td>
</tr>
<tr>
<td>crSTPopStandardDeviation</td>
<td>8</td>
</tr>
<tr>
<td>crSTPopVariance</td>
<td>7</td>
</tr>
<tr>
<td>crSTSampleStandardDeviation</td>
<td>3</td>
</tr>
<tr>
<td>crSTSampleVariance</td>
<td>2</td>
</tr>
<tr>
<td>crSTSum</td>
<td>0</td>
</tr>
</tbody>
</table>

## CRTableDifferences

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crTDOK</td>
<td>0x00000000</td>
</tr>
<tr>
<td>crTDDatabaseNotFound</td>
<td>0x00000001</td>
</tr>
<tr>
<td>crTDServerNotFound</td>
<td>0x00000002</td>
</tr>
<tr>
<td>crTDServerNotOpened</td>
<td>0x00000004</td>
</tr>
<tr>
<td>crTDAliasChanged</td>
<td>0x00000008</td>
</tr>
<tr>
<td>crTDIndexesChanged</td>
<td>0x00000010</td>
</tr>
<tr>
<td>crTDDriverChanged</td>
<td>0x00000020</td>
</tr>
<tr>
<td>crTDDictionaryChanged</td>
<td>0x00000040</td>
</tr>
<tr>
<td>crTDFileTypeChanged</td>
<td>0x00000080</td>
</tr>
<tr>
<td>crTRecordSizeChanged</td>
<td>0x00000100</td>
</tr>
<tr>
<td>crTDAccessChanged</td>
<td>0x00000200</td>
</tr>
<tr>
<td>crTPParametersChanged</td>
<td>0x00000400</td>
</tr>
<tr>
<td>crTLocationChanged</td>
<td>0x00000800</td>
</tr>
<tr>
<td>Constant</td>
<td>Value</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>crTDDatabaseOtherChanges</td>
<td>0x00001000</td>
</tr>
<tr>
<td>crTDNumberFieldChanged</td>
<td>0x00010000</td>
</tr>
<tr>
<td>crTDFieldOtherChanges</td>
<td>0x00020000</td>
</tr>
<tr>
<td>crTDFieldNameChanged</td>
<td>0x00040000</td>
</tr>
<tr>
<td>crTDFieldDescChanged</td>
<td>0x00080000</td>
</tr>
<tr>
<td>crTDFieldTypeChanged</td>
<td>0x00100000</td>
</tr>
<tr>
<td>crTDFieldSizeChanged</td>
<td>0x00200000</td>
</tr>
<tr>
<td>crTDNativeFieldTypeChanged</td>
<td>0x00400000</td>
</tr>
<tr>
<td>crTDNativeFieldOffsetChanged</td>
<td>0x00800000</td>
</tr>
<tr>
<td>crTDNativeFieldSizeChanged</td>
<td>0x01000000</td>
</tr>
<tr>
<td>crTDFieldDecimalPlacesChanged</td>
<td>0x02000000</td>
</tr>
</tbody>
</table>

### CRTextFormat

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crHTMLText</td>
<td>2</td>
</tr>
<tr>
<td>crRTFText</td>
<td>1</td>
</tr>
<tr>
<td>crStandardText</td>
<td>0</td>
</tr>
</tbody>
</table>

### CRTimeBase

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>cr12Hour</td>
<td>0</td>
</tr>
<tr>
<td>cr24Hour</td>
<td>1</td>
</tr>
</tbody>
</table>

### CRTopOrBottomNGroupSortOrder

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crAllGroupsSorted</td>
<td>1</td>
</tr>
<tr>
<td>crAllGroupsUnsorted</td>
<td>0</td>
</tr>
<tr>
<td>crBottomNGroups</td>
<td>3</td>
</tr>
<tr>
<td>crTopNGroups</td>
<td>2</td>
</tr>
</tbody>
</table>
## CRValueFormatType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crAllowComplexFieldFormatting</td>
<td>4</td>
</tr>
<tr>
<td>crIncludeFieldValues</td>
<td>1</td>
</tr>
<tr>
<td>crIncludeHiddenFields</td>
<td>2</td>
</tr>
</tbody>
</table>

## CRViewingAngle

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crBirdsEyeView</td>
<td>15</td>
</tr>
<tr>
<td>crDistortedStdView</td>
<td>10</td>
</tr>
<tr>
<td>crDistortedView</td>
<td>4</td>
</tr>
<tr>
<td>crFewGroupsView</td>
<td>9</td>
</tr>
<tr>
<td>crFewSeriesView</td>
<td>8</td>
</tr>
<tr>
<td>crGroupEmphasisView</td>
<td>7</td>
</tr>
<tr>
<td>crGroupEyeView</td>
<td>6</td>
</tr>
<tr>
<td>crMaxView</td>
<td>16</td>
</tr>
<tr>
<td>crShorterView</td>
<td>12</td>
</tr>
<tr>
<td>crShortView</td>
<td>5</td>
</tr>
<tr>
<td>crStandardView</td>
<td>1</td>
</tr>
<tr>
<td>crTallView</td>
<td>2</td>
</tr>
<tr>
<td>crThickGroupsView</td>
<td>11</td>
</tr>
<tr>
<td>crThickSeriesView</td>
<td>13</td>
</tr>
<tr>
<td>crThickStdView</td>
<td>14</td>
</tr>
<tr>
<td>crTopView</td>
<td>3</td>
</tr>
</tbody>
</table>

## CRYearType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crLongYear</td>
<td>1</td>
</tr>
<tr>
<td>crNoYear</td>
<td>2</td>
</tr>
<tr>
<td>crShortYear</td>
<td>0</td>
</tr>
</tbody>
</table>
What you will find in this chapter...

Print Engine Functions, Page 256
Print Engine Structures, Page 441
Microsoft Windows Structures, Page 523
Print Engine Constants, Page 531
Obsolete Functions, Structures, and Constants, Page 553

...including a list of obsolete calls with applicable replacement calls.
PRINT ENGINE FUNCTIONS

The following functions will be found in this section, arranged alphabetically.

Print Engine Functions A...F, Page 256
Print Engine Functions G (PEGet....), Page 257
Print Engine Functions H...R, Page 259
Print Engine Functions S...Z, Page 260

Print Engine Functions A...F

PEAddParameterCurrentRange, Page 262
PEAddParameterCurrentValue, Page 263
PEAddParameterDefaultValue, Page 264
PECancelPrintJob, Page 265
PECanCloseEngine, Page 266
PECheckFormula, Page 266
PECheckGroupSelectionFormula, Page 267
PECheckNthTableDifferences, Page 268
PECheckSelectionFormula, Page 269
PECheckSQLExpression, Page 270
PEClearParameterCurrentValuesAndRanges, Page 271
PECloseEngine, Page 272
PEClosePrintJob, Page 272
PECloseSubreport, Page 273
PECloseWindow, Page 274
PEConvertPFInfotoVInfo, Page 275
PEConvertVInfotoPFInfo, Page 276
PEDeleteNthGroupSortField, Page 276
PEDeleteNthParameterDefaultValue, Page 278
PEDeleteNthSortField, Page 279
PEDiscardSavedData, Page 280
PEEnableEvent, Page 281
PEEnableProgressDialog, Page 281
Print Engine Functions G (PEGet....)

PEGetAllowPromptDialog, Page 284
PEGetAreaFormat, Page 285
PEGetAreaFormatFormula, Page 286
PEGetEnableEventInfo, Page 287
PEGetErrorCode, Page 288
PEGetErrorText, Page 288
PEGetExportOptions, Page 289
PEGetFieldMappingType, Page 290
PEGetFormula, Page 291
PEGetFormulaSyntax, Page 292
PEGetGraphAxisInfo, Page 293
PEGetGraphFontInfo, Page 294
PEGetGraphOptionInfo, Page 295
PEGetGraphTextDefaultOption, Page 296
PEGetGraphTextInfo, Page 297
PEGetGraphTypeInfo, Page 298
PEGetGroupCondition, Page 299
PEGetGroupOptions, Page 301
PEGetGroupSelectionFormula, Page 302
PEGetHandleString, Page 303
PEGetJobStatus, Page 304
PEGetMargins, Page 305
PEGetNDetailCopies, Page 306
PEGetNFormulas, Page 307
PEGetNGroups, Page 307
PEGetNGroupSortFields, Page 308
PEGetNPages, Page 309
Functions

PEGetNParameterCurrentRanges, Page 310
PEGetNParameterCurrentValues, Page 311
PEGetNParameterDefaultValues, Page 312
PEGetNParameterFields, Page 313
PEGetNSections, Page 313
PEGetNSectionsInArea, Page 314
PEGetNSortFields, Page 315
PEGetNSQLExpressions, Page 315
PEGetNSubreportsInSection, Page 316
PEGetNTables, Page 317
PEGetNthFormula, Page 318
PEGetNthGroupSortField, Page 319
PEGetNthParameterCurrentRange, Page 320
PEGetNthParameterCurrentValue, Page 322
PEGetNthParameterDefaultValue, Page 323
PEGetNthParameterField, Page 324
PEGetNthParameterType, Page 325
PEGetNthParameterValueDescription, Page 326
PEGetNthSortField, Page 327
PEGetNthSQLExpression, Page 328
PEGetNthSubreportInSection, Page 330
PEGetNthTableLocation, Page 331
PEGetNthTableLogOnInfo, Page 332
PEGetNthTablePrivateInfo, Page 333
PEGetNthTableSessionInfo, Page 333
PEGetNthTableType, Page 334
PEGetParameterMinMaxValue, Page 336
PEGetParameterPickListOption, Page 337
PEGetParameterValueInfo, Page 338
PEGetPrintDate, Page 339
Print Engine Functions H...R
PEHasSavedData, Page 357
PEIsPrintJobFinished, Page 358
PELogOffServer, Page 358
PELogOnServer, Page 359
PELogOnSQLServerWithPrivateInfo, Page 361
PENextPrintWindowMagnification, Page 362
PEOpenEngine, Page 363
PEOpenPrintJob, Page 363
PEOpenSubreport, Page 365
PEOutputToPrinter, Page 366
PEOutputToWindow, Page 368
PEPrintControlsShowing, Page 371
Print Engine Functions S...Z

PESelectPrinter, Page 375
PESetAllowPromptDialog, Page 377
PESetAreaFormat, Page 377
PESetAreaFormatFormula, Page 378
PESetDialogParentWindow, Page 380
PESetEventCallback, Page 380
PESetFieldMappingType, Page 388
PESetFont, Page 389
PESetFormula, Page 392
PESetFormulaSyntax, Page 394
PESetGraphAxisInfo, Page 395
PESetGraphFontInfo, Page 396
PESetGraphOptionInfo, Page 397
PESetGraphTextDefaultOption, Page 398
PESetGraphTextInfo, Page 399
PESetGraphTypeInfo, Page 400
PESetGroupCondition, Page 401
PESetGroupOptions, Page 402
PESetGroupSelectionFormula, Page 403
PESetMargins, Page 404
PESetNDetailCopies, Page 406
PESetNthGroupSortField, Page 407
PESetNthParameterDefaultValue, Page 408
PESetNthParameterField, Page 409
PESetNthParameterValueDescription, Page 410
PESetNthSortField, Page 412
PESetNthTableLocation, Page 413
PESetNthTableLogOnInfo, Page 414
PESetNthTablePrivateInfo, Page 415
PESetNthTableSessionInfo, Page 416
PESetParameterMinMaxValue, Page 417
PESetParameterPickListOption, Page 419
PESetParameterValueInfo, Page 420
PESetPrintDate, Page 421
PESetPrintOptions, Page 422
PESetReportOptions, Page 423
PESetReportSummaryInfo, Page 424
PESetReportTitle, Page 424
PESetSectionFormat, Page 426
PESetSectionFormatFormula, Page 427
PESetSectionHeight, Page 428
PESetSelectionFormula, Page 429
PESetSQLExpression, Page 430
PESetSQLQuery, Page 431
PESetTrackCursorInfo, Page 432
PESetWindowOptions, Page 433
PEShow...Page, Page 434
PEShowPrintControls, Page 436
PEStartPrintJob, Page 437
PETestNthTableConnectivity, Page 438
PEVerifyDatabase, Page 439
PEZoomPreviewWindow, Page 440
**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*, Page 21.

**C Syntax**

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job to which you want to add a parameter current range.</td>
</tr>
<tr>
<td>parameterFieldName</td>
<td>Specifies a pointer to the string containing the parameter field name.</td>
</tr>
<tr>
<td>reportName</td>
<td>Specifies a pointer to the string containing the report name. See Remarks below.</td>
</tr>
<tr>
<td>rangeStart</td>
<td>Specifies a pointer to <code>PEValueInfo</code> (page 507), which contains the lower bound of the value range.</td>
</tr>
<tr>
<td>rangeEnd</td>
<td>Specifies a pointer to <code>PEValueInfo</code> (page 507), which contains the upper bound of the value range.</td>
</tr>
<tr>
<td>rangeInfo</td>
<td>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 <code>Range Info Constants</code> (page 551).</td>
</tr>
</tbody>
</table>

**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**

```vbnet
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

```delphi
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, Page 21*.

**C Syntax**

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job to which you want to add a parameter current value.</td>
</tr>
<tr>
<td>parameterFieldName</td>
<td>Specifies a pointer to the string containing the parameter field name.</td>
</tr>
<tr>
<td>reportName</td>
<td>Specifies a pointer to the string containing the report name. See Remarks below.</td>
</tr>
<tr>
<td>currentValue</td>
<td>Specifies a pointer to PEValueInfo (page 507), 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).</td>
</tr>
</tbody>
</table>

**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 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;

PEAddParameterDefaultValu

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job to which you want to add a parameter default value.</td>
</tr>
<tr>
<td>parameterFieldName</td>
<td>Specifies a pointer to the name of the parameter field to which the default value will be added.</td>
</tr>
<tr>
<td>reportName</td>
<td>Specifies a pointer to the report name. See Remarks below.</td>
</tr>
<tr>
<td>valueInfo</td>
<td>Specifies a pointer to PEValueInfo (page 507) which will contain the added default value.</td>
</tr>
</tbody>
</table>

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, Page 436, 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

EXTERNAL CVOID PECancelPrintJob (CWORD) CRPE.DLL
PECanCloseEngine

Use PECanCloseEngine to determine whether or not the Seagate Crystal Report Engine can be closed. Use this function before calling PECloseEngine, Page 272, to verify that the engine is no longer processing print jobs. If the Seagate 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

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

Returns

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

VB Syntax

```vb
Declare Function PECanCloseEngine Lib "crpe32.dll" () As Integer
```

Delphi Syntax

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

dBASE for Windows Syntax

```dBase
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

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job containing the named formula that you want to check.</td>
</tr>
<tr>
<td>formulaName</td>
<td>Specifies a pointer to the name of the formula that you want to check for errors.</td>
</tr>
</tbody>
</table>

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 Seagate Crystal Reports to separate a formula field from other types of fields.
- PECheckFormula works like the Check button that appears in the Seagate 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for the report containing the group selection formula that you want to check.</td>
</tr>
</tbody>
</table>

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 Seagate Crystal Reports Formula Editor.
- If PECheckGroupSelectionFormula returns FALSE, you can provide the associated error text with calls to PEGetErrorCode, Page 288; PEGetErrorText, Page 288; and PEGetHandleString, Page 303.

VB Syntax

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

Delphi Syntax

function PECheckGroupSelectionFormula (printJob: Word): Boolean 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for the report containing the database table that you want to check.</td>
</tr>
<tr>
<td>tableN</td>
<td>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.</td>
</tr>
<tr>
<td>tabledifferenceinfo</td>
<td>Specifies a pointer to PETableDifferenceInfo (page 499), which will contain the information retrieved.</td>
</tr>
</tbody>
</table>
Returns

- TRUE if the call is successful.
- FALSE if the call fails.
- Returns Error Codes (page 537), PE_ERR_NOTIMPLEMENTED if the specified report was based on a dictionary.

VB Syntax

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

Delphi Syntax

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

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

Parameter

```markdown
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job containing the record selection formula that you want to check.</td>
</tr>
</tbody>
</table>
```

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 Seagate Crystal Reports Formula Editor.
- If PECheckSelectionFormula returns FALSE, you can provide the associated error text with calls to PEGetErrorCode, Page 288; PEGetErrorText, Page 288; and PEGetHandleString, Page 303.

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job containing the SQL expression that you want to check.</td>
</tr>
<tr>
<td>expressionName</td>
<td>Specifies a pointer to the name of the SQL expression that you want to check.</td>
</tr>
</tbody>
</table>

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): Boolean 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to clear specified parameter fields.</td>
</tr>
<tr>
<td>parameterFieldName</td>
<td>Specifies a pointer to the string containing the parameter field name.</td>
</tr>
<tr>
<td>reportName</td>
<td>Specifies a pointer to the string containing the report name. See Remarks below</td>
</tr>
</tbody>
</table>

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): Boolean stdcall;
PECloseEngine

Use PECloseEngine to terminate the Seagate Crystal Report Engine. All printing is stopped and all windows are closed. This function stops the Seagate 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 Seagate Crystal Report Engine functions can be called except PEOpenEngine, Page 363.
- Call PECanCloseEngine, Page 266, 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
    cdecl;
```

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 Seagate Crystal Report Engine functions except \( PEOpenPrintJob \), Page 363; \( PECloseEngine \), Page 272; \( PEGetVersion \), Page 353; and \( PELogOnServer \), Page 359, 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, \( PEGetSubreportInfo \), Page 352, or any other Seagate Crystal Report Engine function that applies to the subreport cannot be called.
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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>printJob</code></td>
<td>Specifies the print job for which you want to close the preview window.</td>
</tr>
</tbody>
</table>

Returns

Void.

Remarks

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

VB Syntax

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

Delphi Syntax

```delphi
procedure PECloseWindow ( 
    printJob: Word 
)stdcall;
```
**dBASE for Windows Syntax**

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

**PEConvertPFInfoToVInfo**

Use **PEConvertPFInfoToVInfo** to convert the value returned in the CurrentValue or DefaultValue member of **PEParameterFieldInfo** (page 474), to the appropriate type and place the result in **PEValueInfo** (page 507).

**C Syntax**

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Specifies a pointer to the current or default parameter value (returned from PEGetNthParameterField, Page 324) to convert.</td>
</tr>
<tr>
<td>valueType</td>
<td>Specifies the type of parameter field from which the value came. Use one of the Parameter Field Value Type Constants (page 550).</td>
</tr>
<tr>
<td>valueInfo</td>
<td>Specifies a pointer to <strong>PEValueInfo</strong> (page 507), that is used to store the value in converted form.</td>
</tr>
</tbody>
</table>

**Returns**

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

**VB Syntax**

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

**Delphi Syntax**

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

Use PEConvertVInfotoPFInfo to convert a value contained in PEValueInfo (page 507), into the binary representation expected by PESetNthParameterField, Page 409.

C Syntax

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>valueInfo</td>
<td>Specifies a pointer to PEValueInfo (page 507), that contains the value to be converted.</td>
</tr>
<tr>
<td>valueType</td>
<td>Specifies a pointer to the type of the value.</td>
</tr>
<tr>
<td>value</td>
<td>Specifies a pointer used to store the value in converted form for use in PESetNthParameterField, Page 409.</td>
</tr>
</tbody>
</table>

Returns

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

VB Syntax

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

Delphi Syntax

```delphi
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, Page 308 called once; PEGetNthGroupSortField, Page 319 and PEGetHandleString, Page 303 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

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job from which you want to delete a group sort field.</td>
</tr>
<tr>
<td>sortFieldN</td>
<td>Specifies the 0-based number of the sort field that you want to delete.</td>
</tr>
<tr>
<td></td>
<td>The first group sort field is field 0. If N = 0, the function will delete</td>
</tr>
<tr>
<td></td>
<td>the first group sort field. If the report has N group sort fields, the</td>
</tr>
<tr>
<td></td>
<td>function can be called with sortFieldN between 0 and N-1.</td>
</tr>
</tbody>
</table>

Returns

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

Remarks

This function should be called before PEStartPrintJob, Page 437 or the results may be inconsistent or unexpected.

VB Syntax

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

Delphi Syntax

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

dBASE for Windows Syntax

```dBASE
EXTERN CLOGICAL PEDeleteNthGroupSortField (CWORD, CWORD) CRPE.DLL
```
**C/VB/Delphi Functions**

**PEDeleteNthParameterDefaultValue**

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

**C Syntax**

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job from which you want to delete a default value.</td>
</tr>
<tr>
<td>parameterFieldName</td>
<td>Specifies a pointer to the string containing the parameter field name.</td>
</tr>
<tr>
<td>reportName</td>
<td>Specifies a pointer to a string containing the report name. See Remarks below.</td>
</tr>
<tr>
<td>index</td>
<td>Specifies the index number of the default value to be deleted.</td>
</tr>
</tbody>
</table>

**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**

```vbnet
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**

```delphi
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, Page 315, called once; PEGetNthSortField, Page 327 and PEGetHandleString, Page 303 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**

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

**Parameters**

<table>
<thead>
<tr>
<th>printJob</th>
<th>Specifies the print job from which you want to delete a sort field.</th>
</tr>
</thead>
<tbody>
<tr>
<td>sortFieldN</td>
<td>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.</td>
</tr>
</tbody>
</table>

**Returns**

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

**Remarks**

This function should be called before PES tmPrintJob, Page 437, or the results may be inconsistent or unexpected.

**VB Syntax**

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

**Delphi Syntax**

```delphi
function PEDeleteNthSortField ( 
    printJob: Word; 
    sortFieldN: integer 
): Bool stdcall;
```
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 Seagate Crystal Report Engine to retrieve new data the next time the report is printed.

C Syntax

```c
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, Page 437, or the results may be inconsistent or unexpected.

VB Syntax

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

Delphi Syntax

```delphi
function PEDiscardSavedData (  
    printJob: Word  
): BOOL stdcall;
```

dBASE for Windows Syntax

```dBASE
EXTERN CLOGICAL PEDiscardSavedData (CWORD, CWORD) CRPE.DLL
```
PEEnableEvent

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

**C Syntax**

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

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to enable or disable events.</td>
</tr>
<tr>
<td>enableEventInfo</td>
<td>Specifies a pointer to PEEnableEventInfo (page 446).</td>
</tr>
</tbody>
</table>

**Returns**

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

**Delphi Syntax**

```delphi
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**

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

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to enable/disable the progress dialog box.</td>
</tr>
<tr>
<td>enable</td>
<td>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.</td>
</tr>
</tbody>
</table>
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, Page 436, disables the Print Control buttons.

C Syntax

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job that you want to export to a disk file or to e-mail.</td>
</tr>
<tr>
<td>toMail</td>
<td>Boolean value indicates whether or not the function is to export to e-mail.</td>
</tr>
<tr>
<td>waitUntilDone</td>
<td>BOOL. Reserved. This parameter must always be set to TRUE.</td>
</tr>
</tbody>
</table>

Returns

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

Declare Function PEExportPrintWindow Lib "crpe32.d11" ( 
  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 (page 448). 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job that you want to export.</td>
</tr>
<tr>
<td>options</td>
<td>Specifies a pointer to PEExportOptions (page 448).</td>
</tr>
</tbody>
</table>

Returns

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

VB Syntax

Declare Function PEExportTo Lib "crpe32.d11" ( 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: PEEExportOptions
): 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, Page 375, or PEGetSelectedPrinter, Page 347.

C Syntax

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to release the specified DEVMODE (page 523), Windows API structure.</td>
</tr>
<tr>
<td>mode</td>
<td>Specifies a pointer to the DEVMODE (page 523), Windows API structure that you want to release.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to determine whether prompting for parameter values is allowed during printing.</td>
</tr>
</tbody>
</table>

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 (page 491). Use this function to update the area formats and pass them back using PESetAreaFormat, Page 377.

C Syntax

BOOL CRPE_API PEGetAreaFormat (    short printJob,    short areaCode,    PESectionOptions FAR *options );

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job from which you want to get the area format.</td>
</tr>
<tr>
<td>areaCode</td>
<td>Specifies the Section Codes (page 551), for the area for which you want to get format information. See Working with section codes, Page 22.</td>
</tr>
<tr>
<td>options</td>
<td>Specifies a pointer to PESectionOptions (page 491), which will receive format information for the area.</td>
</tr>
</tbody>
</table>

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, Page 378. Use PEGetHandleString, Page 303, to retrieve the formula text.

C Syntax

BOOL CRPE_API PEGetAreaFormatFormula ( 
    short printJob,
    short areaCode,
    short formulaName,
    HANDLE FAR *textHandle,
    short FAR *textLength );

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job from which you want to get the area format formula.</td>
</tr>
<tr>
<td>areaCode</td>
<td>Specifies the Section Codes (page 551), for the area for which you want to get format information. See Working with section codes, Page 22.</td>
</tr>
<tr>
<td>formulaName</td>
<td>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 (page 533).</td>
</tr>
<tr>
<td>textHandle</td>
<td>Specifies a pointer to the handle of the string containing the formula text.</td>
</tr>
<tr>
<td>textLength</td>
<td>Specifies a pointer to the length of the formula string (in bytes) including the terminating byte.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to obtain information about enabled events.</td>
</tr>
<tr>
<td>enableEventInfo</td>
<td>Specifies a pointer to PEEnableEventInfo (page 446).</td>
</tr>
</tbody>
</table>

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 Seagate 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

```c
short CRPE_API PEGetErrorCode ( 
    short printJob );
```

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>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.</td>
</tr>
</tbody>
</table>

Returns

- Returns the Error Codes (page 537), of the most recent error for the given job if the function was unsuccessful.
- Returns 0 if the function completed without error.

VB Syntax

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

Delphi Syntax

```delphi
function PEGetErrorCode ( 
    printJob: Word 
): Smallint stdcall;
```

dBASE for Windows Syntax

```dBASE
EXTERN CWORD PEGetErrorCode (CWORD) CRPE.DLL
```

PEGetErrorText

Use PEGetErrorText to return a string handle describing the status of the most recent Seagate Crystal Report Engine function called. This function is used with PEGetHandleString, Page 303. These functions can be used in a Custom-Print Link to display the error string as part of an error message.
C Syntax

```c
BOOL CRPE_API PEGetErrorText ( 
    short printJob, 
    HANDLE FAR *textHandle, 
    short Far *textLength );
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>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.</td>
</tr>
<tr>
<td>textHandle</td>
<td>Specifies a pointer to the handle of the string containing the error text.</td>
</tr>
<tr>
<td>textLength</td>
<td>Specifies a pointer to the length of the error string (in bytes) including the terminating byte.</td>
</tr>
</tbody>
</table>

**Returns**

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

VB Syntax

```vb
Declare Function PEGetErrorText Lib “crpe32.dll” (ByVal printJob As Integer, 
    TextHandle As Long, TextLength As Integer) As Integer
```

Delphi Syntax

```delphi
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 Seagate Crystal Report Engine to fill in PEExportOptions (page 448). Function PEEExportTo, Page 283, can then be used to set the print job destination using the information in PEExportOptions (page 448).

C Syntax

```c
BOOL CRPE_API PEGetExportOptions ( 
    short printJob, 
    PEEXportOptions FAR *options );
```
## Functions

### PEGetExportOptions

Use PEGetExportOptions to retrieve the export options for a specified report.

**VB Syntax**

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

**Delphi Syntax**

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job from which you want to get export options.</td>
</tr>
<tr>
<td>options</td>
<td>Specifies a pointer to <strong>PEExportOptions</strong> (page 448).</td>
</tr>
</tbody>
</table>

**Returns**

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

### PEGetFieldMappingType

Use PEGetFieldMappingType to retrieve the field mapping type code for the specified report.

**C Syntax**

```c
BOOL CRPE_API PEGetFieldMappingType (
    short printJob,
    WORD FAR *mappingType );
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job from which the field mapping type code will be retrieved.</td>
</tr>
<tr>
<td>mappingType</td>
<td>Specifies a pointer to the appropriate <strong>PE_FM_XXX Field Mapping Type Constants</strong> (page 542).</td>
</tr>
</tbody>
</table>

**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, Page 303. Use PESetFormula, Page 392, 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job from which you want to retrieve the formula string.</td>
</tr>
<tr>
<td>formulaName</td>
<td>Specifies a pointer to the null-terminated string that contains the name of the formula for which you want to retrieve the formula string.</td>
</tr>
<tr>
<td>textHandle</td>
<td>Specifies a pointer to the handle of the string containing the formula text.</td>
</tr>
<tr>
<td>textLength</td>
<td>Specifies a pointer to the length of the formula string (in bytes) including the terminating byte.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>printJob</th>
<th>Specifies the print job for which you want to determine formula syntax.</th>
</tr>
</thead>
<tbody>
<tr>
<td>formulaSyntax</td>
<td>Specifies a pointer to PEFormulaSyntax (page 456), which will contain the information that you want to retrieve. See Remarks below.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job from which you want to retrieve chart axis information.</td>
</tr>
<tr>
<td>sectionN</td>
<td>Specifies the section of the report containing the chart for which you want to retrieve chart axis information.</td>
</tr>
<tr>
<td>graphN</td>
<td>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.</td>
</tr>
<tr>
<td>graphAxisInfo</td>
<td>Specifies a pointer to PEGraphAxisInfo (page 458), which will contain the information retrieved.</td>
</tr>
</tbody>
</table>

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**

```c
BOOL CRPE_API PEGetGraphFontInfo (
    short printJob,
    short sectionN,
    short graphN,
    WORD titleFontType,
    PEFontColorInfo FAR * fontColourInfo );
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to retrieve chart font information.</td>
</tr>
<tr>
<td>sectionN</td>
<td>Specifies the number of the section in which the chart appears. This parameter should be within the range obtained by PEGetNSections, Page 313.</td>
</tr>
<tr>
<td>graphN</td>
<td>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.</td>
</tr>
<tr>
<td>titleFontType</td>
<td>Uses one of the PE_GTF_XXX Graph Text Font Constants (page 546).</td>
</tr>
<tr>
<td>fontColourInfo</td>
<td>Specifies a pointer to PEFontColorInfo, which will contain the information retrieved.</td>
</tr>
</tbody>
</table>

**Returns**

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

**VB Syntax**

```vba
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**

```delphi
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

```c
BOOL CRPE_API PEGetGraphOptionInfo ( 
    short printJob, 
    short sectionN, 
    short graphN, 
    PEGraphOptionInfo FAR * graphOptionInfo );
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to retrieve chart display information.</td>
</tr>
<tr>
<td>sectionN</td>
<td>Specifies the number of the section in which the chart appears. This parameter should be within the range obtained by <strong>PEGetNSections</strong>, Page 313.</td>
</tr>
<tr>
<td>graphN</td>
<td>Specifies for which chart within the Charts are numbered based on their order of insertion into the report.</td>
</tr>
<tr>
<td>graphOptionInfo</td>
<td>Specifies a pointer to <strong>PEGraphOptionInfo</strong> (page 461), which will contain the retrieved information.</td>
</tr>
</tbody>
</table>

#### Returns

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

### VB Syntax

```vb
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

```delphi
function PEGetGraphOptionInfo ( 
    printJob            : Smallint; 
    sectionN            : Smallint; 
    graphN              : Smallint; 
    var graphOptionInfo : PEGraphOptionInfo): Bool; 
{$ifdef WIN32} stdcall; {$endif}
```

---

_Seagate Crystal Report Engine_ 295
PEGetGraphTextDefaultOption

Use PEGetGraphTextDefaultOption to determine whether or not default chart titles will be displayed.

**C Syntax**

```c
BOOL CRPE_API PEGetGraphTextDefaultOption (
    short printJob,
    short sectionN,
    short graphN,
    WORD titleType,
    BOOL FAR *useDefault );
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to retrieve chart title default option information.</td>
</tr>
<tr>
<td>sectionN</td>
<td>Specifies the 0-based index number of the section in which the chart appears. This parameter should be within the range obtained by PEGetNSections, Page 313.</td>
</tr>
<tr>
<td>graphN</td>
<td>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.</td>
</tr>
<tr>
<td>titleType</td>
<td>Specifies the title type. Use one of the PE_GTT_XXX Graph Title Type Constants (page 546).</td>
</tr>
<tr>
<td>useDefault</td>
<td>Specifies a pointer to the Boolean value indicating whether or not to default chart title will be displayed.</td>
</tr>
</tbody>
</table>

**Returns**

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

**VB Syntax**

```vb
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, Page 303.

C Syntax

```c
BOOL CRPE_API PEGetGraphTextInfo (  
    short printJob,  
    short sectionN,  
    short graphN,  
    WORD titleType,  
    HANDLE FAR *title,  
    short FAR *titleLength );
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to retrieve title text information.</td>
</tr>
<tr>
<td>sectionN</td>
<td>Specifies the number of the section in which the chart appears. This parameter should be within the range obtained by PEGetNSections, Page 313.</td>
</tr>
<tr>
<td>graphN</td>
<td>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.</td>
</tr>
<tr>
<td>titleType</td>
<td>Specifies the title type. Use one of the PE_GTT_XXX Graph Title Type Constants (page 546).</td>
</tr>
<tr>
<td>title</td>
<td>Specifies a pointer to the handle of the title.</td>
</tr>
<tr>
<td>titleLength</td>
<td>Specifies a pointer to the length of the title.</td>
</tr>
</tbody>
</table>

Returns

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

VB Syntax

```vbnet
Declare Function PEGetGraphTextInfo Lib "crpe32.dll" (  
    ByVal printJob As Integer, ByVal sectionN As Integer,  
    ByVal graphN As Integer, ByVal titleType As Integer,  
    ByVal title As Long, ByVal titleLength As Integer ) As Integer
```
Delphi Syntax

```delphi
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

```c
BOOL CRPE_API PEGetGraphTypeInfo(
    short printJob,
    short sectionN,
    short graphN,
    PEGraphTypeInfo FAR * graphTypeInfo);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to retrieve chart type information.</td>
</tr>
<tr>
<td>sectionN</td>
<td>Specifies the number of the section in which the chart appears. This parameter should be within the range obtained by PEGetNSections, Page 313.</td>
</tr>
<tr>
<td>graphN</td>
<td>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.</td>
</tr>
<tr>
<td>graphTypeInfo</td>
<td>Specifies a pointer to PEGraphTypeInfo (page 464), which will contain the retrieved information.</td>
</tr>
</tbody>
</table>

Returns

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

VB Syntax

```vb
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

```delphi
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`, Page 401, to change the group condition once it is known.

C Syntax

```c
BOOL CRPE_API PEGetGroupCondition (  
  short printJob,  
  short sectionCode,  
  HANDLE FAR *conditionFieldHandle,  
  short FAR *conditionFieldLength,  
  short FAR *condition,  
  short FAR *sortDirection );
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job that you want to query to determine the group conditions for a selected group.</td>
</tr>
<tr>
<td>sectionCode</td>
<td>Specifies the Section Codes (page 551), for the report section for which you want to get the group condition. See Working with section codes, Page 22.</td>
</tr>
<tr>
<td>conditionFieldHandle</td>
<td>Specifies a pointer to the handle of the condition field for the selected group section.</td>
</tr>
<tr>
<td>conditionFieldLength</td>
<td>Specifies a pointer to the length of the condition field for the selected group section.</td>
</tr>
<tr>
<td>condition</td>
<td>Specifies a pointer to the condition setting for the selected group section. See Remarks below.</td>
</tr>
<tr>
<td>sortDirection</td>
<td>Specifies a pointer to the sort direction setting for the selected group section. Uses one of the PE_SF_XXX Sort Order Constants (page 551).</td>
</tr>
</tbody>
</table>

**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:

```c
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**

```c
BOOL CRPE_API PEGetGroupOptions (
    short printJob,
    short groupN,
    PEGroupOptions FAR *groupOptions );
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to obtain information about the group options settings.</td>
</tr>
<tr>
<td>groupN</td>
<td>Specifies the 0-based group number.</td>
</tr>
<tr>
<td>groupOptions</td>
<td>Specifies a pointer to <strong>PEGroupOptions</strong> (page 465).</td>
</tr>
</tbody>
</table>

**Returns**

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

**Remarks**

This function gets all the information retrieved with **PEGetGroupCondition**, Page 299, plus additional group options such as repeat group header, keep group together, and top/bottom n group sorting.

**VB Syntax**

```vb
Declare Function PEGetGroupOptions Lib "crpe32.dll" (ByVal printJob As Integer, ByVal groupN As Integer, groupOptions As PEGroupOptions) As Integer
```

**Delphi Syntax**

```delphi
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, Page 303. Use PESetGroupSelectionFormula, Page 403, 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

```c
BOOL CRPE_API PEGetGroupSelectionFormula(
    short printJob,
    HANDLE FAR *textHandle,
    short FAR *textLength);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to retrieve the group selection formula string.</td>
</tr>
<tr>
<td>textHandle</td>
<td>Specifies a pointer to the handle of the string containing the formula text.</td>
</tr>
<tr>
<td>textLength</td>
<td>Specifies a pointer to the length of the formula string (in bytes) including the terminating byte.</td>
</tr>
</tbody>
</table>

Returns

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

VB Syntax

```vb
Declare Function PEGetGroupSelectionFormula Lib "crpe32.dll" (ByVal printJob As Integer, TextHandle As Long, TextLength As Integer) As Integer
```

Delphi Syntax

```delphi
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

```c
BOOL CRPE_API PEGetHandleString(
    HANDLE textHandle,
    char FAR *buffer,
    short bufferLength);
```

Parameters

<table>
<thead>
<tr>
<th>textHandle</th>
<th>Specifies the handle of the string containing the text of interest. This handle is obtained from a variable length string function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer</td>
<td>Specifies a pointer to the buffer into which you want the string copied.</td>
</tr>
<tr>
<td>bufferLength</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

```vb
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 PEStrartPrintJob, Page 437, returns.

C Syntax

short CRPE_API PEGetJobStatus (  
    short printJob,  
    PEJobInfo FAR *jobInfo );

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to determine printing status.</td>
</tr>
<tr>
<td>jobInfo</td>
<td>Specifies a pointer to PEJobInfo (page 468), which will contain the information this function retrieves.</td>
</tr>
</tbody>
</table>

Returns

- Returns 0 if PEOpenEngine or PEOpenPrintJob has not been called successfully.
- Otherwise, returns the Job Status Constants (page 549) 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

```delphi
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, Page 404, to set the report margins.

C Syntax

```c
BOOL CRPE_API PEGetMargins ( 
  short printJob,
  short FAR *left,
  short FAR *right,
  short FAR *top,
  short FAR *bottom );
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job that you want to query to retrieve margin information.</td>
</tr>
<tr>
<td>left</td>
<td>Specifies a pointer to the value of the left margin.</td>
</tr>
<tr>
<td>right</td>
<td>Specifies a pointer to the value of the right margin.</td>
</tr>
<tr>
<td>top</td>
<td>Specifies a pointer to the value of the top margin.</td>
</tr>
<tr>
<td>bottom</td>
<td>Specifies a pointer to the value of the bottom margin.</td>
</tr>
</tbody>
</table>

Returns

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

VB Syntax

```vb
Declare Function PEGetMargins Lib "crpe32.dll" ( ByVal printJob As Integer,
    LeftMargin As Integer, RightMargin As Integer,
    TopMargin As Integer, BottomMargin As Integer ) As Integer
```
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, Page 406.

C Syntax

```c
BOOL CRPE_API PEGetNDetailCopies (
    short printJob,
    short FAR *nDetailCopies );
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job that you want to query to determine how many times each Details section is to be printed.</td>
</tr>
<tr>
<td>nDetailCopies</td>
<td>Specifies a pointer to the number of copies of the Details section to be printed.</td>
</tr>
</tbody>
</table>

Returns

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

VB Syntax

```vb
Declare Function PEGetNDetailCopies Lib "crpe32.d11" (
    ByVal printJob As Integer, nDetailCopies As Integer ) As Integer
```

Delphi Syntax

```delphi
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`, Page 318.

**C Syntax**

```c
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**

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

**Delphi Syntax**

```delphi
function PEGetNFormulas(printJob: Word): Smallint stdcall;
```

**dBASE for Windows Syntax**

```dBASE
EXTERN CWORD PEGetNFormulas (CWORD) CRPE.DLL
```

**PEGetNGroups**

Use PEGetNGroups to determine the number of groups in the specified report.

**C Syntax**

```c
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, 
Page 319 and PEGetHandleString, Page 303 called as many times as needed to identify the correct group sort
field; and PESetNthGroupSortField, Page 407 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, Page 434), for example.

C Syntax

short CRPE_API PEGetNPages ( short printJob );

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to determine the number of pages.</td>
</tr>
</tbody>
</table>

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, Page 21*.

**C Syntax**

```c
short CRPE_API PEGetNParameterCurrentRanges ( 
    short printJob, 
    const char FAR *parameterFieldName, 
    const char FAR *reportName );
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to retrieve parameter current range information.</td>
</tr>
<tr>
<td>parameterFieldName</td>
<td>Specifies a pointer to the string containing the parameter field name.</td>
</tr>
<tr>
<td>reportName</td>
<td>Specifies a pointer to the string containing the report name. See Remarks below.</td>
</tr>
</tbody>
</table>

**Returns**

- Returns the number of value ranges associated with the specified parameter field.
- Returns -1 if an error occurs.

**VB Syntax**

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

**Delphi Syntax**

```delphi
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, Page 21.

**C Syntax**

```c
short CRPE_API PEGetNParameterCurrentValues (
    short printJob,
    const char FAR *parameterFieldName,
    const char FAR *reportName );
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>printJob</code></td>
<td>Specifies the print job for which you want to determine the number of parameter current values.</td>
</tr>
<tr>
<td><code>parameterFieldName</code></td>
<td>Specifies a pointer to the string containing the parameter field name.</td>
</tr>
<tr>
<td><code>reportName</code></td>
<td>Specifies a pointer to the string containing the report name. See Remarks below.</td>
</tr>
</tbody>
</table>

**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**

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

**Delphi Syntax**

```delphi
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

```c
short CRPE_API PEGetNParameterDefaultValues (
    short printJob
    const char FAR *parameterFieldName
    const char FAR *reportName);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to determine the number of parameter default values.</td>
</tr>
<tr>
<td>parameterFieldName</td>
<td>Specifies a pointer to the string containing the name of the parameter field.</td>
</tr>
<tr>
<td>reportName</td>
<td>Specifies a pointer to the string containing the name of the report. See Remarks below.</td>
</tr>
</tbody>
</table>

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

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

Delphi Syntax

```delphi
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**

```c
short CRPE_API PEGetNParameterFields (short printJob);
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job from which you want to retrieve a parameter field count.</td>
</tr>
</tbody>
</table>

**Returns**

- The number of parameter fields in the report.
- Returns -1 if an error occurs.

**VB Syntax**

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

**Delphi Syntax**

```delphi
function PEGetNParameterFields (printJob: Word): 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**

```c
short CRPE_API PEGetNSections (short printJob);
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job from which you want to get a section count.</td>
</tr>
</tbody>
</table>
Returns
- Returns the number of sections in the report.
- Returns -1 if an error occurs.

VB Syntax
Declare Function PEGetNSections Lib "crpe32.d11" (  
    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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to determine the number of sections in the specified area.</td>
</tr>
<tr>
<td>areaCode</td>
<td>Specifies the area for which you want to retrieve the section count.</td>
</tr>
</tbody>
</table>

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.d11" (  
    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; PEGetNthSortField, Page 327 and PEGetHandleString, Page 303 called together as many times as needed to identify the correct sort field; and PESetNthSortField, Page 412 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to determine the number of sort fields contained.</td>
</tr>
</tbody>
</table>

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 );
### Functions

#### Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>print job</td>
<td>Specifies the print job for which you want to determine the number of SQL expressions.</td>
</tr>
</tbody>
</table>

#### 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 determine the number of subreports in the specified section.

#### C Syntax

```
short CRPE_API PEGetNSubreportsInSection (short printJob, short sectionCode);
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>print job</td>
<td>Specifies the primary report from which you want to retrieve information about the number of subreports in a section.</td>
</tr>
<tr>
<td>sectionCode</td>
<td>Specifies the Section Codes (page 551), of the section for which you want a subreport count. See Working with section codes, Page 22.</td>
</tr>
</tbody>
</table>

#### Returns

- Returns the number of subreports in the specified section.
- Returns -1 if an error occurs.

#### Remarks

sectionCode can be retrieved using PEGetSectionCode, Page 343.
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**

```delphi
function PEGetNTables (  
    printJob: Word  
): smallint stdcall;
```

**dBASE for Windows Syntax**

```c
EXTERN CWORD PEGetNTables (CWORD) CRPE.DLL
```

**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`, Page 392.

**C Syntax**

```c
BOOL CRPE_API PEGetNthFormula (  
    short printJob,  
    short formulaN,  
    HANDLE FAR *nameHandle,  
    short FAR *nameLength,  
    HANDLE FAR *textHandle,  
    short FAR *textLength );
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>printJob</code></td>
<td>Specifies the print job from which you want to gather formula information.</td>
</tr>
<tr>
<td><code>formulaN</code></td>
<td>Specifies the 0-based number of the formula about which you want to gather information.</td>
</tr>
<tr>
<td><code>nameHandle</code></td>
<td>Specifies a pointer to the handle of the string containing the formula name.</td>
</tr>
<tr>
<td><code>nameLength</code></td>
<td>Specifies a pointer to the length of the formula name string (in bytes) including the terminating byte.</td>
</tr>
<tr>
<td><code>textHandle</code></td>
<td>Specifies a pointer to the handle of the string containing the formula text.</td>
</tr>
<tr>
<td><code>textLength</code></td>
<td>Specifies a pointer to the length of the formula string (in bytes) including the terminating byte.</td>
</tr>
</tbody>
</table>

### Returns

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

---

318
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, Page 303. PEGetNthGroupSortField returns the name of the 
field and the direction (ascending or descending) of the sort. See Crystal Report Engine API variable length 
strings, Page 27 for additional information.

This function is typically used as one of a series of functions (PEGetNGroupSortFields, Page 308) called once; 
PEGetNthGroupSortField and PEGetHandleString, Page 303 called as many times as needed to identify the 
correct group sort field; and PESetNthGroupSortField, Page 407 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job from which you want to gather group sort field information.</td>
</tr>
<tr>
<td>sortFieldN</td>
<td>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.</td>
</tr>
<tr>
<td>nameHandle</td>
<td>Specifies a pointer to the handle of the string containing the sort field name.</td>
</tr>
</tbody>
</table>
Returns

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

Remarks

To find out top/bottom n group sort information, use `PEGetGroupOptions`, Page 310.

**VB Syntax**

```vbnet
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**

```delphi
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`, Page 310, to get the number of value ranges currently associated with the parameter field. See Working with Parameter Values and Ranges, Page 21.

**C Syntax**

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to retrieve the parameter current range.</td>
</tr>
<tr>
<td>parameterFieldName</td>
<td>Specifies a pointer to the string containing the parameter field name.</td>
</tr>
<tr>
<td>reportName</td>
<td>Specifies a pointer to the string containing the report name. See Remarks below.</td>
</tr>
<tr>
<td>index</td>
<td>Specifies the index number of the value range to be retrieved.</td>
</tr>
<tr>
<td>rangeStart</td>
<td>Specifies a pointer to <code>PEValueInfo</code> (page 507), in which the beginning value in the range is returned.</td>
</tr>
<tr>
<td>rangeEnd</td>
<td>Specifies a pointer to <code>PEValueInfo</code> (page 507), in which the final value in the range is returned.</td>
</tr>
<tr>
<td>rangeInfo</td>
<td>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 <code>Range Info Constants</code> (page 551).</td>
</tr>
</tbody>
</table>

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

```vbnet
Declare Function PEGetNthParameterCurrentRange Lib "crpe32.d11" ( _
    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

```delphi
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, Page 311, to determine the number of values currently held in the parameter field. See Working with Parameter Values and Ranges, Page 21.

C Syntax

```c
BOOL CRPE_API PEGetNthParameterCurrentValue ( 
    short printJob, 
    const char FAR *parameterFieldame, 
    const char FAR *reportName, 
    short index, 
    PEValueInfo FAR *currentValue );
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to determine the parameter current value.</td>
</tr>
<tr>
<td>parameterName</td>
<td>Specifies a pointer to a string containing the parameter field name.</td>
</tr>
<tr>
<td>reportName</td>
<td>Specifies a pointer to a string containing the report name. See Remarks below.</td>
</tr>
<tr>
<td>index</td>
<td>Specifies the index number of the value to be retrieved.</td>
</tr>
<tr>
<td>currentValue</td>
<td>Specifies a pointer to <code>PEValueInfo</code> (page 507), in which the value will be returned. If it contains no value then it will be set to the constant <code>PE_VI_NOVALUE</code> rather than NULL.</td>
</tr>
</tbody>
</table>

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

```vb
Declare Function PEGetNthParameterCurrentValue Lib "crpe32.d11" ( _
    ByVal printJob As Integer, ByVal parameterFieldName As String, _
    ByVal reportName As String, ByVal index As Integer, _
    currentValue As PEValueInfo ) As Integer
```
Delphi Syntax

```delphi
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 PEGetNthParameterDefaultValues, Page 312, to retrieve the number of default values for the parameter field.

C Syntax

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job from which you want to gather parameter default value information.</td>
</tr>
<tr>
<td>parameterFieldName</td>
<td>Specifies a pointer to the string containing the parameter field name.</td>
</tr>
<tr>
<td>reportName</td>
<td>Specifies a pointer to the string containing the report name. See Remarks below.</td>
</tr>
<tr>
<td>index</td>
<td>Specifies the index number of the default value to be retrieved.</td>
</tr>
<tr>
<td>valueInfo</td>
<td>Specifies a pointer to PEValueInfo (page 507), containing information about requested default value.</td>
</tr>
</tbody>
</table>

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, Page 313, (called once); PEGetNthParameterField (called as many times as needed to identify the correct parameter field); and PESetNthParameterField, Page 409, (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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job that contains the parameter field about which you want to retrieve information.</td>
</tr>
<tr>
<td>parameterN</td>
<td>Specifies the number of the parameter field about which you want to retrieve information.</td>
</tr>
<tr>
<td>parameterInfo</td>
<td>Specifies a pointer to PEParameterFieldInfo (page 474), which is used to store the information you retrieve. See Remarks below.</td>
</tr>
</tbody>
</table>

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, Page 310
  — PEGetNParameterCurrentValues, Page 311
  — PEGetNParameterDefaultValues, Page 312
  — PEGetNthParameterCurrentRange, Page 320
  — PEGetNthParameterCurrentValue, Page 322
  — PEGetNthParameterDefaultValue, Page 323
● The CurrentValue member of the returned structure, PEParameterFieldInfo (page 474), 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
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job from which you want to gather parameter type information.</td>
</tr>
<tr>
<td>index</td>
<td>Specifies the index number of the parameter field.</td>
</tr>
</tbody>
</table>
Returns

Returns one of the following PE_PO_XXX Constants or -1 if the index is invalid.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_PO_REPORT</td>
<td>Report</td>
</tr>
<tr>
<td>PE_PO_STOREDPROC</td>
<td>Stored Procedure</td>
</tr>
<tr>
<td>PE_PO_QUERY</td>
<td>Query</td>
</tr>
</tbody>
</table>

VB Syntax

Declare Function PEGetNthParameterType Lib "crpe32.d11" ( _
    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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job from which you want to gather parameter value description information.</td>
</tr>
<tr>
<td>parameterFieldName</td>
<td>Specifies a pointer to the parameterFieldName for which you want to retrieve the parameter value description.</td>
</tr>
<tr>
<td>reportName</td>
<td>Specifies a pointer to the report name. See Remarks below.</td>
</tr>
<tr>
<td>index</td>
<td>Specifies the index.</td>
</tr>
<tr>
<td>valueDesc</td>
<td>Specifies a pointer to the handle of the value description to be retrieved.</td>
</tr>
<tr>
<td>valueDescLength</td>
<td>Specifies a pointer to the length of the value description.</td>
</tr>
</tbody>
</table>
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.d11" (_
    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}

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,
Page 315 called once; PEGetNthSortField and PEGetHandleString, Page 303 called together as many times as
needed to identify the correct sort field; and PESetNthSortField, Page 412 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job from which you want to retrieve sort field information.</td>
</tr>
<tr>
<td>sortFieldN</td>
<td>Specifies the 0-based number of the sort field you want to retrieve. The first sort field is</td>
</tr>
<tr>
<td></td>
<td>field 0. If the report has N sort fields, the function can be called with sortFieldN between</td>
</tr>
<tr>
<td></td>
<td>0 and N-1.</td>
</tr>
<tr>
<td>nameHandle</td>
<td>Specifies a pointer to the handle of the string containing the sort field name.</td>
</tr>
<tr>
<td>nameLength</td>
<td>Specifies a pointer to the length of the field name string (in bytes) including the terminating</td>
</tr>
<tr>
<td></td>
<td>byte.</td>
</tr>
<tr>
<td>direction</td>
<td>Specifies a pointer to the sort direction. Uses one of the PE_SF_XXX Sort Order Constants</td>
</tr>
<tr>
<td></td>
<td>(page 551).</td>
</tr>
</tbody>
</table>

Returns

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

VB Syntax

```vbnet
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

```delphi
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, Page 303. Use PEGetNSQLExpressions, Page 315, to determine the number of SQL expressions in the report.
C Syntax

```c
BOOL CRPE_API PEGetNthSQLExpression (  
    short printJob,  
    short expressionN,  
    HANDLE FAR *nameHandle,  
    short FAR *nameLength,  
    HANDLE FAR *textHandle,  
    short FAR *textLength );
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job from which you want to gather SQL expression information.</td>
</tr>
<tr>
<td>expressionN</td>
<td>Specifies the numeric value indicating which expression to retrieve.</td>
</tr>
<tr>
<td>nameHandle</td>
<td>Specifies a pointer to the handle of the string containing the expression name.</td>
</tr>
<tr>
<td>nameLength</td>
<td>Specifies a pointer to the length of name string.</td>
</tr>
<tr>
<td>textHandle</td>
<td>Specifies a pointer to the handle of the string containing the SQL expression.</td>
</tr>
<tr>
<td>textLength</td>
<td>Specifies a pointer to the length of the expression string.</td>
</tr>
</tbody>
</table>

Returns

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

VB Syntax

```vb
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

```delphi
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

```c
DWORD CRPE_API PEGetNthSubreportInSection ( 
    short printJob, 
    short sectionCode, 
    short subreportN );
```

Parameters

<table>
<thead>
<tr>
<th>printJob</th>
<th>Specifies the primary report.</th>
</tr>
</thead>
<tbody>
<tr>
<td>sectionCode</td>
<td>Specifies the <strong>Section Codes</strong> (page 551), for the report section that contains the subreport. See <em>Working with section codes, Page 22</em>.</td>
</tr>
<tr>
<td>subreportN</td>
<td>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.</td>
</tr>
</tbody>
</table>

Returns

Returns a handle that is used to retrieve the name of the specified subreport.

Remarks

Use PEGetSubreportInfo, Page 352, to retrieve information about the subreport by passing the subreport handle returned by PEGetNthSubreportInSection.

VB Syntax

```vbnet
Declare Function PEGetNthSubreportInSection Lib "crpe32.dll" ( 
    ByVal printJob As Integer, ByVal sectionCode As Integer, 
    ByVal subreportN As Integer ) As Long
```

Delphi Syntax

```delphi
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, Page 413 to identify the location of a table and then to change it.

C Syntax

```c
BOOL CRPE_API PEGetNthTableLocation (
    short printJob,
    short tableN,
    PETableLocation FAR *location );
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job from which you want to retrieve information about a table's location.</td>
</tr>
<tr>
<td>tableN</td>
<td>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.</td>
</tr>
<tr>
<td>location</td>
<td>Specifies the pointer to PETableLocation (page 501). The format of the string will be depend on the type of database specified.</td>
</tr>
</tbody>
</table>

Returns

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

VB Syntax

```vb
Declare Function PEGetNthTableLocation Lib "crpe32.dll" ( ByVal printJob As Integer, ByVal TableN As Integer, Location As PETableLocation ) As Integer
```

Delphi Syntax

```delphi
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

```c
BOOL CRPE_API PEGetNthTableLogOnInfo (
    short printJob,
    short tableN,
    PELogOnInfo FAR *logOnInfo );
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to get table log on information.</td>
</tr>
<tr>
<td>tableN</td>
<td>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.</td>
</tr>
<tr>
<td>logOnInfo</td>
<td>Specifies the pointer to PELogOnInfo (page 470).</td>
</tr>
</tbody>
</table>

Returns

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

Remarks

- This function must be called after PEOpenPrintJob, Page 363 (so you have the job handle), and before PEStartPrintJob, Page 437 (which needs the password set to print the report).
- The password in PELogOnInfo (page 470) will always be an empty string when using this function.

VB Syntax

```vb
Declare Function PEGetNthTableLogOnInfo Lib "crpe32.dll" ( ByVal printJob As Integer, ByVal TableN As Integer, LogOnInfo As PELogOnInfo ) As Integer
```

Delphi Syntax

```delphi
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**

```c
BOOL CRPE_API PEGetNthTablePrivateInfo (  
    short printJob,  
    short tableN,  
    PETablePrivateInfo FAR *privateInfo );
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Identifies the print job which uses the table for which you want to retrieve the session information.</td>
</tr>
<tr>
<td>tableN</td>
<td>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.</td>
</tr>
<tr>
<td>privateInfo</td>
<td>Specifies the pointer to PETablePrivateInfo (page 502).</td>
</tr>
</tbody>
</table>

**Returns**

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

**Delphi Syntax**

```delphi
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**

```c
BOOL CRPE_API PEGetNthTableSessionInfo (  
    short printJob,  
    short tableN,  
    PESessionInfo FAR *sessionInfo );
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Identifies the print job that uses the table for which you want to retrieve the session information.</td>
</tr>
<tr>
<td>tableN</td>
<td>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.</td>
</tr>
<tr>
<td>sessionInfo</td>
<td>Specifies a pointer to PESessionInfo (page 493).</td>
</tr>
</tbody>
</table>

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 (page 493) will always be an empty string when using this function.

VB Syntax

```vbnet
Declare Function PEGetNthTableSessionInfo Lib "crpe32.dll" ( ByVal printJob As Integer, ByVal TableN As Integer, SessionInfo As PESessionInfo ) As Integer
```

Delphi Syntax

```delphi
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

```delphi
BOOL CRPE_API PEGetNthTableType ( short printJob, short tableN, PETableType FAR *tableType );
```
Seagate Crystal Report Engine

Functions

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to determine a table type.</td>
</tr>
<tr>
<td>tableN</td>
<td>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.</td>
</tr>
<tr>
<td>tableType</td>
<td>Specifies a pointer to PETableType (page 503).</td>
</tr>
</tbody>
</table>

Returns

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

Remarks

- The application can test DBType in PETableType (page 503) 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 PEStrartPrintJob.

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

```c
BOOL CRPE_API PEGetParameterMinMaxValue ( 
    short printJob, 
    const char FAR *parameterFieldName, 
    const char FAR *reportName, 
    PEValueInfo FAR *valueMin, 
    PEValueInfo FAR *valueMax );
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to retrieve the parameter min/max values.</td>
</tr>
<tr>
<td>parameterFieldName</td>
<td>Specifies a pointer to the string containing the parameter name.</td>
</tr>
<tr>
<td>reportName</td>
<td>Specifies a pointer to the string containing the report name. See Remarks below.</td>
</tr>
<tr>
<td>valueMin</td>
<td>Specifies a pointer to PEValueInfo (page 507), in which information about the minimum parameter value will be returned. See Remarks below.</td>
</tr>
<tr>
<td>valueMax</td>
<td>Specifies a pointer to PEValueInfo (page 507), in which information about the maximum parameter value will be returned. See Remarks below.</td>
</tr>
</tbody>
</table>

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` (page 478).

**C Syntax**

```
BOOL CRPE_API PEGetParameterPickListOption ( 
  short printJob, 
  const char FAR *parameterFieldName, 
  const char FAR *reportName, 
  PEParameterPickListOption FAR *pickListOption );
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job from which you want to retrieve the parameter pick list options.</td>
</tr>
<tr>
<td>parameterFieldName</td>
<td>Specifies a pointer to the parameterFieldName for which you want to retrieve pick list options.</td>
</tr>
<tr>
<td>reportName</td>
<td>Specifies a pointer to the report name. See Remarks below.</td>
</tr>
<tr>
<td>pickListOption</td>
<td>Specifies a pointer to <code>PEParameterPickListOption</code> (page 478), which will contain the information retrieved.</td>
</tr>
</tbody>
</table>

**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 and 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 (page 479), 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, Page 21.

C Syntax

BOOL CRPE_API PEGetParameterValueInfo (short printJob,
const char FAR *parameterFieldName,
const char FAR *reportName,
PEParameterValueInfo FAR *valueInfo );

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job from which you want to gather parameter value information.</td>
</tr>
<tr>
<td>parameterFieldName</td>
<td>Specifies a pointer to the string containing the parameter field name.</td>
</tr>
<tr>
<td>reportName</td>
<td>Specifies a pointer to the string containing the report name. See Remarks below.</td>
</tr>
<tr>
<td>valueInfo</td>
<td>Specifies a pointer to PEParameterValueInfo (page 479), in which the parameter value information will be returned. See Working with Parameter Values and Ranges, Page 21.</td>
</tr>
</tbody>
</table>
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 reportName : 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, Page 421.

C Syntax

BOOL CRPE_API PEGetPrintDate ( 
    short printJob, 
    short FAR *year, 
    short FAR *month, 
    short FAR *day );

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to retrieve the print date setting.</td>
</tr>
<tr>
<td>year</td>
<td>Specifies a pointer to the year component of the print date.</td>
</tr>
<tr>
<td>month</td>
<td>Specifies a pointer to the month component of the print date.</td>
</tr>
<tr>
<td>day</td>
<td>Specifies a pointer to the day component of the print date.</td>
</tr>
</tbody>
</table>
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
): Boolean 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 (page 480). Use this function to retrieve print options from the report in order to update them and pass back using PESetPrintOptions, Page 422.

C Syntax

BOOL CRPE_API PEGetPrintOptions (  
  short printJob,
  PEPrintOptions FAR *options );

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job that you want to query to determine which print options have been set using the Print common dialog box.</td>
</tr>
<tr>
<td>options</td>
<td>Specifies a pointer to PEPrintOptions (page 480).</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to determine report options.</td>
</tr>
<tr>
<td>reportOptions</td>
<td>Specifies a pointer to PEReportOptions (page 485), which contains report options for the specified print job.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to get the report summary information.</td>
</tr>
<tr>
<td>summaryInfo</td>
<td>Specifies a pointer to PEReportSummaryInfo (page 488).</td>
</tr>
</tbody>
</table>

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, Page 303. Use PESetReportTitle, Page 424, 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 );
### Functions

#### VB/Delphi

#### PEGetReportTitle

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to get the report title.</td>
</tr>
<tr>
<td>titleHandle</td>
<td>Specifies a pointer to the handle of the title string.</td>
</tr>
<tr>
<td>titleLength</td>
<td>Specifies a pointer to the length of the title string (in bytes) including the terminating byte.</td>
</tr>
</tbody>
</table>

#### Returns

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

**VB Syntax**

```vbscript
Declare Function PEGetReportTitle Lib "crpe32.dll" ( ByVal printJob As Integer, TitleHandle As Long, TitleLength As Integer ) As Integer
```

**Delphi Syntax**

```delphi
function PEGetReportTitle ( printJob: Word; var titleHandle: HWnd; var titleLength: Word ): 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*, Page 22.

**C Syntax**

```c
short CRPE_API PEGetSectionCode ( short printJob, short sectionN );
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job from which you want to retrieve a section code.</td>
</tr>
<tr>
<td>sectionN</td>
<td>Specifies the number of the section for which you want the section code. This parameter should be within the range obtained by PEGetNSections, Page 313.</td>
</tr>
</tbody>
</table>
Returns

- Returns the Section Codes (page 551), 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 (page 491). Use this function in order to update the section formats and pass them back using PESetSectionFormat, Page 426.

C Syntax

BOOL CRPE_API PEGetSectionFormat (  
    short printJob,  
    short sectionCode,  
    PESectionOptions FAR *options );

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job that you want to query to determine what report section options have been set using the Format Section dialog box.</td>
</tr>
<tr>
<td>sectionCode</td>
<td>Specifies the Section Codes (page 551), for the report section(s) for which you want to get section format information. See Working with section codes, Page 22.</td>
</tr>
<tr>
<td>options</td>
<td>Specifies a pointer to PESectionOptions (page 491).</td>
</tr>
</tbody>
</table>

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, Page 303.

C Syntax

BOOL CRPE_API PEGetSectionFormatFormula (  
    short printJob,  
    short sectionCode,  
    short formulaName,  
    HANDLE FAR *textHandle,  
    short FAR *textLength );

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to retrieve the section format formula.</td>
</tr>
<tr>
<td>sectionCode</td>
<td>Specifies the section codes (page 551), for the report section(s) for which you want to get section format information. See Working with section codes, Page 22.</td>
</tr>
<tr>
<td>formulaName</td>
<td>Specifies the name of the section format formula. Use one of the PE_FFN_XXX Area/Section Format Formula Constants (page 533).</td>
</tr>
<tr>
<td>textHandle</td>
<td>Specifies a pointer to the handle of the text of the actual formula.</td>
</tr>
<tr>
<td>textLength</td>
<td>Specifies a pointer to the length of the text string. Use this value to allocate a buffer for the text.</td>
</tr>
</tbody>
</table>

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 \textit{PEGetHandleString, Page 303}, to fill the buffer with the actual text of the formula.

VB Syntax

\begin{verbatim}
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
\end{verbatim}

Delphi Syntax

\begin{verbatim}
function PEGetSectionFormatFormula ( 
    printJob: Word; 
    sectionCode: Word; 
    formulaName: Word; 
    var textHandle: HWnd; 
    var textLength: Word 
): Bool stdcall;
\end{verbatim}

\textit{PEGetSectionHeight}

Use \textit{PEGetSectionHeight} to retrieve the section height information for the specified section. This is the replacement API Call for \textit{PEGetMinimumSectionHeight} and should be used for all new development.

C Syntax

\begin{verbatim}
BOOL CRPE_API PEGetSectionHeight ( 
    short printJob, 
    short sectionCode, 
    short FAR *height );
\end{verbatim}

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job that you want to query to retrieve section height information.</td>
</tr>
<tr>
<td>sectionCode</td>
<td>Specifies the \textbf{Section Codes} (page 551), for the report sections for which you want to retrieve information. See Working with section codes, Page 22.</td>
</tr>
<tr>
<td>height</td>
<td>Specifies a pointer to the section height (in twips).</td>
</tr>
</tbody>
</table>
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 Seagate 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  
        DEVMODEA FAR * FAR *mode  
    #endif
);
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job that you want to query to get information on the non-default printer that has been selected with the report.</td>
</tr>
<tr>
<td>driverHandle</td>
<td>Specifies a pointer to the handle of the printer driver for the printer that is selected with the print job.</td>
</tr>
<tr>
<td>driverLength</td>
<td>Specifies a pointer to the length of the printer driver name.</td>
</tr>
<tr>
<td>printerHandle</td>
<td>Specifies a pointer to the handle of the printer that is selected with the print job.</td>
</tr>
<tr>
<td>printerLength</td>
<td>Specifies a pointer to the length of the printer name.</td>
</tr>
<tr>
<td>portHandle</td>
<td>Specifies a pointer to the handle of the port to which the selected printer is connected.</td>
</tr>
<tr>
<td>portLength</td>
<td>Specifies a pointer to the length of the port name.</td>
</tr>
<tr>
<td>mode</td>
<td>Specifies a pointer to the DEVMODE (page 523), or DEVMODE Windows API structure.</td>
</tr>
</tbody>
</table>

Returns

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

Remarks

- Use PEGetHandleString, Page 303, to obtain the actual strings pointed to by the string handles returned.

VB Syntax

```vbnet
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

```delphi
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**, Page 303. Use **PESetSelectionFormula**, Page 429, 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**

```c
BOOL CRPE_API PEGetSelectionFormula (
    short printJob,
    HANDLE FAR *textHandle,
    short FAR *textLength );
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to retrieve the selection formula string.</td>
</tr>
<tr>
<td>textHandle</td>
<td>Specifies a pointer to the handle of the string containing the formula text.</td>
</tr>
<tr>
<td>textLength</td>
<td>Specifies a pointer to the length of the formula string (in bytes) including the terminating byte.</td>
</tr>
</tbody>
</table>

**Returns**

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

**VB Syntax**

```vbnet
Declare Function PEGetSelectionFormula Lib "crpe32.dll" (ByVal printJob As Integer, TextHandle As Long, TextLength As Integer) As Integer
```

**Delphi Syntax**

```delphi
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, Page 303.

C Syntax

```c
BOOL CRPE_API PEGetSQLExpression (  
    short printJob,  
    const char FAR *expressionName,  
    HANDLE FAR *textHandle,  
    short FAR *textLength );
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to retrieve the SQL expression string.</td>
</tr>
<tr>
<td>expressionName</td>
<td>Specifies a pointer to the expression name.</td>
</tr>
<tr>
<td>textHandle</td>
<td>Specifies a pointer to the handle of the expression string.</td>
</tr>
<tr>
<td>textLength</td>
<td>Specifies a pointer to the length of the expression string.</td>
</tr>
</tbody>
</table>

Returns

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

VB Syntax

```vb
Declare Function PEGetSQLExpression Lib "crpe32.dll" (ByVal printJob As Integer, ByVal expressionName As String, textHandle As Long, textLength As Integer) As Integer
```

Delphi Syntax

```delphi
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 Seagate Crystal Reports, in a syntax that’s specific to the database driver you’re using. Use this function with PEGetHandleString, Page 303. Use PESetSQLQuery, Page 431, to update the query. See Remarks below.

C Syntax

```c
BOOL CRPE_API PEGetSQLQuery (  
    short printJob,  
    HANDLE FAR *textHandle,  
    short FAR *textLength );
```

Parameters

<table>
<thead>
<tr>
<th>printJob</th>
<th>Specifies the print job from which you want to retrieve the SQL query.</th>
</tr>
</thead>
<tbody>
<tr>
<td>textHandle</td>
<td>Specifies a pointer to the handle of the string containing the SQL query string.</td>
</tr>
<tr>
<td>textLength</td>
<td>Specifies a pointer to the length of the SQL query string (in bytes) including the terminating byte.</td>
</tr>
</tbody>
</table>

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

```vb
Declare Function PEGetSQLQuery Lib “crpe32.dll” ( ByVal printJob As Integer,  
TextHandle As Long, TextLength As Integer ) As Integer
```

Delphi Syntax

```delphi
function PEGetSQLQuery (  
  printJob: Word;  
  var textHandle: HWnd;  
  var textLength: Word  
): Bool stdcall;
```
**PEGetSubreportInfo**

Use PEGetSubreportInfo to retrieve information about the specified subreport.

**C Syntax**

```c
BOOL CRPE_API PEGetSubreportInfo (
    short printJob,
    DWORD subreportHandle,
    PESubreportInfo FAR *subreportInfo );
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the primary report that contains the subreport about which you want to retrieve information.</td>
</tr>
<tr>
<td>subreportHandle</td>
<td>Specifies the handle of the subreport about which you want to retrieve information.</td>
</tr>
<tr>
<td>subreportInfo</td>
<td>Specifies a pointer to <strong>PESubreportInfo</strong> (page 498), which will be used for holding the information once it is retrieved.</td>
</tr>
</tbody>
</table>

**Returns**

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

**VB Syntax**

```vbnet
Declare Function PEGetSubreportInfo Lib "crpe32.d11" (  
    ByVal printJob As Integer, ByVal subreportHandle As Long,  
    subreportInfo As PESubreportInfo ) As Integer
```

**Delphi Syntax**

```delphi
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

```c
BOOL CRPE_API PEGetTrackCursorInfo (
    short printJob,
    PETrackCursorInfo FAR *cursorInfo );
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to retrieve track cursor information.</td>
</tr>
<tr>
<td>cursorInfo</td>
<td>Specifies a pointer to PETrackCursorInfo (page 504), which will contain the track cursor information.</td>
</tr>
</tbody>
</table>

Returns

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

Delphi Syntax

```delphi
function PEGetTrackCursorInfo ( 
    printJob: smallint; 
    var cursorInfo: PETrackCursorInfo 
): Bool stdcall;
```

PEGetVersion
Use PEGetVersion to retrieve the version number of the DLL or the Seagate 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 Seagate 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 Seagate Crystal Report Engine with the older version earlier in the path than the new version.

C Syntax

```c
short CRPE_API PEGetVersion ( 
    short versionRequested );
```
### C/VB/Delphi Functions

#### Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>versionRequested</td>
<td>Specifies whether the DLL or Seagate Crystal Report Engine version is being requested. Use one of the following PE_GV_XXX constants.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_GV_DLL</td>
<td>Returns the version of the DLL (CRPE/CRPE32).</td>
</tr>
<tr>
<td>PE_GV_ENGINE</td>
<td>Returns the version of the Seagate Crystal Report Engine.</td>
</tr>
</tbody>
</table>

#### Returns

Returns the version number of the DLL or the Seagate Crystal Report Engine.

### VB Syntax

```vbnet
Declare Function PEGetVersion Lib "crpe32.dll" (ByVal version As Integer) As Integer
```

The following simplifies the PEGetVersion call in Visual Basic.

```vbnet
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

```delphi
function PEGetVersion (versionRequested: integer): Smallint stdcall;
```

### dBASE for Windows Syntax

```dbase
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**

```c
HWND CRPE_API PEGetWindowHandle (
    short printJob );
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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`, Page 437, and then, only if you have created a preview window.

**VB Syntax**

```vb
Declare Function PEGetWindowHandle Lib "crpe32.dll" ( 
    ByVal printJob As Integer ) As Long
```

**Delphi Syntax**

```delphi
function PEGetWindowHandle ( 
    printJob: Word 
): HWND stdcall;
```

**dBASE for Windows Syntax**

```dBASE
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**

```c
BOOL CRPE_API PEGetWindowOptions (
    short printJob,
    PEWindowOptions FAR *options );
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job from which you want to retrieve the preview window</td>
</tr>
<tr>
<td></td>
<td>options. If two or more preview windows are open, this function applies only to</td>
</tr>
<tr>
<td></td>
<td>the most recently created preview window.</td>
</tr>
<tr>
<td>PEWindowOptions</td>
<td>Specifies a pointer to <strong>PEWindowOptions</strong> (page 509), which will contain the</td>
</tr>
<tr>
<td></td>
<td>retrieved information.</td>
</tr>
</tbody>
</table>

**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 Seagate 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**

```vbnet
Declare Function PEGetWindowOptions Lib "crpe32.d11" ( 
    ByVal printJob As Integer, Options As PEWindowOptions ) As Integer
```

**Delphi Syntax**

```delphi
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

```c
BOOL CRPE_API PEHasSavedData ( 
    short printJob,
    BOOL FAR *hasSavedData );
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the handle of the print job you want to query to determine if it has saved data with it.</td>
</tr>
<tr>
<td>hasSavedData</td>
<td>Specifies a pointer to a Boolean value that indicates whether or not there is data saved with the report.</td>
</tr>
</tbody>
</table>

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, Page 280, 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

```vb
Declare Function PEHasSavedData Lib "crpe32.dll" ( 
    ByVal printJob As Integer, HasSavedData As Long ) As Integer
```

Delphi Syntax

```delphi
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

```c
BOOL CRPE_API PEIsPrintJobFinished (short printJob);
```

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job that you want to query to determine if it has finished printing.</td>
</tr>
</tbody>
</table>

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

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

Delphi Syntax

```delphi
function PEIsPrintJobFinished (printJob: Word): Bool stdcall;
```

dBASE for Windows Syntax

```dBASE
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

```c
BOOL CRPE_API PELogOffServer (const char FAR *dllName, PELogOnInfo FAR *logOnInfo);
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dllName</td>
<td>Specifies a pointer to the name of the Seagate 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.</td>
</tr>
<tr>
<td>logOnInfo</td>
<td>Specifies a pointer to PELogOnInfo (page 470).</td>
</tr>
</tbody>
</table>

Returns

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

Remarks

- PELogOnServer, Page 359, 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, Page 414, was already used to logon to a table.
- This function requires a database DLL name which can be retrieved using PEGetNthTableType, Page 334.

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dllName</td>
<td>Specifies a pointer to the name of the Seagate Crystal Reports DLL for the server or password-protected non-SQL table to which you want to logon. (for example, &quot;PDSODBC.DLL&quot;). 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.</td>
</tr>
<tr>
<td>logOnInfo</td>
<td>Specifies a pointer to PELogOnInfo (page 470).</td>
</tr>
</tbody>
</table>

Returns

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

Remarks

- PELogOnServer and PELogOffServer, Page 358, can be called at any time to log on and off a database server. These functions are not required if PESetNthTableLogOnInfo, Page 414, was already used to set the password for a table.
- This function requires a database DLL name, which can be retrieved using PEGetNthTableType, Page 334.
- 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 PESetNthTableLogOnInfo, Page 364, to perform logon instead.
- The following points need to be considered when deciding whether to use PELogOnServer or PESetNthTableLogOnInfo.
  - PELogOnServer is easier to call than PESetNthTableLogOnInfo, Page 414, 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 PESetNthTableLogOnInfo.

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 Seagate 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 Seagate Crystal Reports Log Off call from disconnecting an application’s existing connection to the Server.

C Syntax

```c
BOOL CRPE_API PELogOnSQLServerWithPrivateInfo (  
    const char FAR *dllName,  
    void FAR *privateInfo );
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dllName</td>
<td>Specifies a pointer to the name of the Seagate 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 Seagate Crystal Reports DLL is PDSODBC.DLL.</td>
</tr>
<tr>
<td>privateInfo</td>
<td>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 Seagate 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.</td>
</tr>
</tbody>
</table>

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

```vb
Declare Function PELogOnSQLServerWithPrivateInfo Lib "crpe32.dll" ( ByVal DLLName As String, ByVal PrivateInfo As Long ) As Integer
```
Delphi Syntax

```delphi
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**

```c
BOOL CRPE_API PENextPrintWindowMagnification (  
short printJob  
);
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>printJob</code></td>
<td>Specifies the print job displayed in the preview window for which you want to step the magnification level.</td>
</tr>
</tbody>
</table>

**Returns**

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

**VB Syntax**

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

**Delphi Syntax**

```delphi
function PENextPrintWindowMagnification (  
printJob: Word  
): Bool stdcall;
```

**dBASE for Windows Syntax**

```dbase
EXTERN CLOGICAL PENextPrintWindowMagnification (CWORD) CRPE.DLL
```
PEOpenEngine

Use PEOpenEngine to prepare the Seagate 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**

```c
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 Seagate Crystal Report Engine function. If an error occurs in the PEOpenEngine function call, `PEGGetErrorCode, Page 288`, 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**

```vb
Declare Function PEOpenEngine Lib "crpe32.dll" () As Integer
```

**Delphi Syntax**

```delphi
function PEOpenEngine : Bool stdcall;
```

**dBASE for Windows Syntax**

```dbase
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**

```c
short PEOpenPrintJob (const char *reportFilePath);
```
**C/VB/Delphi**

**Functions**

**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.
- PEOpenPrintJob, Page 272, 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**

```vbnet
Declare Function PEOpenPrintJob Lib "crpe32.dll" (ByVal RptName As String) As Integer
```

**Delphi Syntax**

```delphi
function PEOpenPrintJob (reportFilePath: PChar): Smallint stdcall;
```

**dBASE for Windows Syntax**

```sql
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

```c
short CRPE_API PEOpenSubreport (
    short parentJob,
    char FAR *subreportName );
```

Parameters

<table>
<thead>
<tr>
<th>parentJob</th>
<th>Specifies the primary report (the report that contains the subreport). This is the handle returned from PEOpenPrintJob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>subreportName</td>
<td>Specifies a pointer to the name of the subreport that you want to open. This is retrieved using PEGetSubreportInfo, Page 352.</td>
</tr>
</tbody>
</table>

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

```vb
Declare Function PEOpenSubreport Lib "crpe32.dll" ( 
    ByVal parentJob As Integer, ByVal subreportName As String ) As Integer
```

Delphi Syntax

```delphi
function PEOpenSubreport ( 
    parentJob: Word;
    subreportName: PChar
): Word stdcall;
```
PEOutputToPrinter

Use PEOutputToPrinter to direct output to a printer. See Remarks below.

C Syntax

```c
BOOL CRPE_API PEOutputToPrinter ( 
    short printJob, 
    short nCopies );
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job that you want to send to a printer.</td>
</tr>
<tr>
<td>nCopies</td>
<td>Specifies the number of report copies that you want to print. Pass 0 to preserve the existing setting.</td>
</tr>
</tbody>
</table>

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, Page 375, 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, Page 363, 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

```c
// 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.
— PESetPrintJob
   // 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.
— PESetPrintJob
   // 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 PEOuputToDefaultPrinter 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

```c
BOOL CRPE_API PEOutputToWindow( 
    short printJob, 
    const char FAR *title, 
    int left, 
    int top, 
    int width, 
    int height, 
    DWORD style, 
    HWND parentWindow );
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job you want to print in the preview window.</td>
</tr>
<tr>
<td>title</td>
<td>Specifies a pointer to the null-terminated string that contains the title that you want to appear in the preview window title bar.</td>
</tr>
<tr>
<td>left</td>
<td>Specifies the x coordinate of the upper left corner of the preview window, in pixels. See Remarks below.</td>
</tr>
<tr>
<td>top</td>
<td>Specifies the y coordinate of the upper left corner of the preview window, in pixels. See Remarks below.</td>
</tr>
<tr>
<td>width</td>
<td>Specifies the width of the preview window, in pixels.</td>
</tr>
<tr>
<td>height</td>
<td>Specifies the height of the preview window, in pixels.</td>
</tr>
<tr>
<td>style</td>
<td>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.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WS_MINIMIZE</td>
<td>536870912</td>
<td>Make a window of minimum size.</td>
</tr>
<tr>
<td>WS_VISIBLE</td>
<td>268435456</td>
<td>Make a window that is visible when it first appears (for overlapping and pop-up windows).</td>
</tr>
<tr>
<td>WS_DISABLED</td>
<td>134217728</td>
<td>Make a window that is disabled when it first appears.</td>
</tr>
<tr>
<td>WS_CLIPSIBLINGS</td>
<td>67108864</td>
<td>Clip child windows with respect to one another.</td>
</tr>
</tbody>
</table>
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_DLGFFRAME 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_USEDEFAULT -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_USEDEFAULT 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, Page 437`, is called.

**VB Syntax**

```vbnet
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**

```delphi
function PEOutputToWindow (printJob: Word; title: PChar; left: longint; top: longint; width: longint; height: longint; style: longint; parentWindow: HWnd): Bool stdcall;
```

**dBASE for Windows Syntax**

```dbase
EXTERN CLOGICAL PEOutputToWindow (CWORD, CSTRING, CWORD, CWORD, CWORD, CWORD, CWORD, CLOG, 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, Page 436.

C Syntax

```c
BOOL CRPE_API PEPrintControlsShowing(
    short printJob,
    BOOL FAR *controlsShowing);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>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.</td>
</tr>
<tr>
<td>controlsShowing</td>
<td>Specifies a pointer to a TRUE value if the print controls will be shown or FALSE value if they will be hidden.</td>
</tr>
</tbody>
</table>

Returns

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

VB Syntax

```vb
Declare Function PEPrintControlsShowing Lib "crpe32.dll" ( ByVal printJob As Integer, ControlsShowing As Long ) As Integer
```

Delphi Syntax

```delphi
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

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reportFilePath</td>
<td>Specifies a pointer to the NULL-terminated string that contains the name and path of the report that you want to print.</td>
</tr>
<tr>
<td>toDefaultPrinter</td>
<td>Specifies whether or not the report is to be sent to the default printer.</td>
</tr>
<tr>
<td>toWindow</td>
<td>Specifies whether or not the report is to be displayed in the preview window.</td>
</tr>
<tr>
<td>title</td>
<td>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.</td>
</tr>
<tr>
<td>left</td>
<td>Specifies the x coordinate of the upper left hand corner of the preview window, in pixels.</td>
</tr>
<tr>
<td>top</td>
<td>Specifies the y coordinate of the upper left hand corner of the preview window, in pixels.</td>
</tr>
<tr>
<td>width</td>
<td>Specifies the width of the preview window, in pixels.</td>
</tr>
<tr>
<td>height</td>
<td>Specifies the height of the preview window, in pixels.</td>
</tr>
<tr>
<td>style</td>
<td>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 (page 368).</td>
</tr>
<tr>
<td>parentWindow</td>
<td>Specifies the handle of the parent window if the preview window is a child of that window.</td>
</tr>
</tbody>
</table>
Returns

- Returns PE_ERR_NOERROR if the call was successful.
- Returns another Error Codes (page 537), 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job displayed in the preview window that you want to print.</td>
</tr>
<tr>
<td>waitUntilDone</td>
<td>BOOL. Reserved. This parameter must always be set to TRUE.</td>
</tr>
</tbody>
</table>
Returns

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

**VB Syntax**

```vbnet
Declare Function PEPrintWindow Lib "crpe32.dll" (  ByVal printJob As Integer, ByVal WaitNoWait As Integer ) As Integer
```

**Delphi Syntax**

```delphi
function PEPrintWindow (  printJob: Word;  waitUntilDone: Bool  ): Bool stdcall;
```

**dBASE for Windows Syntax**

```dBASE
EXTERN CLOGICAL PEPrintWindow (CWORD, CLOGICAL) CRPE.DLL
```

**PEReimportSubreport**

Use PEReimportSubreport to reimport a subreport into the specified main report.

**C Syntax**

```c
BOOL CRPE_API PEReimportSubreport (  short printJob,  DWORD subreportHandle,  BOOL FAR *linkChanged,  BOOL FAR *reimported );
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to reimport a subreport.</td>
</tr>
<tr>
<td>subreportHandle</td>
<td>Specifies the handle of the subreport that you want to reimport.</td>
</tr>
<tr>
<td>linkChanged</td>
<td>Specifies a pointer to a Boolean value indicating whether or not the link has changed. See Remarks below.</td>
</tr>
<tr>
<td>reimported</td>
<td>Specifies a pointer to a Boolean value indicating whether or not the subreport has been reimported. See Remarks below.</td>
</tr>
</tbody>
</table>

**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, Page 288.

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 (page 523), 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 PEOoutputToPrinter, Page 366) or to the preview window (using PEOoutputToWindow, Page 368).
- This call will override a printer selection that you built into the report at design time via the Seagate Crystal Reports Select Printer menu command.
- If you follow this call with the call PEOoutputToWindow, Page 368, 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 (page 523), structure to customize (if your development tool supports such a structure).
- This function should be called before PESStartPrintJob, Page 437, or the results may be inconsistent or unexpected.

**VB Syntax**

```vba
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**

```delphi
function PESelectPrinter(  
  printJob: Word;  
  driverName: PChar;  
  printerName: PChar;  
  portName: PChar;  
  mode: PDeviceModeA  
): Bool stdcall;
```
PESetAllowPromptDialog

Use PESetAllowPromptDialog to specify whether prompting for parameter values is allowed during printing.

C Syntax

```c
BOOL CRPE_API PESetAllowPromptDialog (  
    short printJob,  
    BOOL showPromptDialog );
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job.</td>
</tr>
<tr>
<td>showPromptDialog</td>
<td>If TRUE, then prompting is allowed.</td>
</tr>
</tbody>
</table>

Returns

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

VB Syntax

```vb
Declare Function PESetAllowPromptDialog Lib "crpe32.dll" (ByVal printJob As Integer, ByVal showPromptDialog As Integer) As Integer
```

Delphi Syntax

```delphi
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 (page 491). 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

```c
BOOL CRPE_API PESetAreaFormat (  
    short printJob,  
    short areaCode,  
    PESectionOptions FAR *options );
```
Parameters

<table>
<thead>
<tr>
<th>printJob</th>
<th>Specifies the print job for which you want to set area formatting options.</th>
</tr>
</thead>
<tbody>
<tr>
<td>areaCode</td>
<td>Specifies the Section Codes (page 551) for the report area for which you want to set formatting options. See the information on area codes in Working with section codes, Page 22.</td>
</tr>
<tr>
<td>options</td>
<td>Specifies a pointer to PESectionOptions (page 491). Use this structure to set your section options.</td>
</tr>
</tbody>
</table>

Returns

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

Remarks

This function should be called before PESetPrintJob, Page 437, 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to set a new format formula string.</td>
</tr>
<tr>
<td>areaCode</td>
<td>Specifies the Section Codes (page 551) for the report area for which you want to set formatting options. See Working with section codes, Page 22.</td>
</tr>
<tr>
<td>formulaName</td>
<td>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 (page 533).</td>
</tr>
<tr>
<td>formulaString</td>
<td>Specifies a pointer to the null-terminated string that you want to assign to the format formula.</td>
</tr>
</tbody>
</table>

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, Page 437, 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**

```c
BOOL CRPE_API PESetDialogParentWindow (short printJob, HWND parentWindow);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to specify a parent window.</td>
</tr>
<tr>
<td>parentWindow</td>
<td>Specifies the handle of the parent window.</td>
</tr>
</tbody>
</table>

**Returns**

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

**VB Syntax**

```vbnet
Declare Function PESetDialogParentWindow Lib "crpe32.dll" (ByVal printJob As Integer, ByVal parentWindow As Long) As Integer
```

**Delphi Syntax**

```delphi
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**

```c
BOOL CRPE_API PESetEventCallback (short printJob, 
    BOOL (CALLBACK *callbackProc) (short eventID, void *param, void *userData)
    void *userData);
```
Parameters

<table>
<thead>
<tr>
<th>printJob</th>
<th>Specifies the print job for which you want to create an Event callback procedure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>callbackProc</td>
<td>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.</td>
</tr>
<tr>
<td>userData</td>
<td>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.</td>
</tr>
</tbody>
</table>

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, Page 35.
- 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 (page 456).

- 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` (page 456).

- **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` (page 443).

- **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` (page 456).

- **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` (page 456).

- **Return**
  Ignored.

PE_DRILL_ON_DETAIL_EVENT

- **Called**
  After double-clicking one of the detail areas in the preview window.
**PE_DRILL_ON_DETAIL_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 `PEDrillOnDetailEventInfo` (page 443).

- **Return**
  Ignored.

**PE_DRILL_ON_GROUP_EVENT**

- **Called**
  Whenever the user double-clicked on an object with a hyperlink (and the hyperlink is about to be executed).

- **Parameter**
  Pointer to `PEDrillOnGroupEventInfo` (page 445).

- **Return**
  TRUE to use the default action; FALSE to cancel the default action.

**PE_DRILL_ON_HYPERLINK_EVENT**

- **Called**
  After clicking the first page button; before going to the first page.

- **Parameter**
  Pointer to `PEGeneralPrintWindowEventInfo` (page 456).

- **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 (page 456).
- 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 (page 456).
- 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.
- Parameter
  Pointer to PELaunchSeagateAnalysisEventInfo (page 470).
- 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 (page 472).
- 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, Page 290, and PESetFieldMappingType, Page 388.
- Parameter
  Pointer to PEFIELDMappingEventInfo (page 452).
● 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** (page 472).

- **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** (page 456).

- **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** (page 456).

- **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** (page 456).

- **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** (page 456).
● 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 (page 482).

● Return
   Ignored.

PE_REFRESH_BUTTON_CLICKED_EVENT

● Called
   After clicking the Refresh button; before refreshing the data.

● Parameter
   Pointer to PEGeneralPrintWindowEventInfo (page 456).

● 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 (page 472).

● 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 (page 490).

● 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** (page 495).

● **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** (page 496).

● **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** (page 497).

● **Return**
  Ignored.

**PE_ZOOM_LEVEL_CHANGING_EVENT**

● **Called**
  After changing zoom control; before changing preview zoom level.

● **Parameter**
  Pointer to **PEZoomLevelChangingEventInfo** (page 511).

● **Return**
  Ignored.

**Delphi Syntax**

```
function PESetEventCallback(
  printJob: Word;
  callbackProc: pointer
  {Callback Function should be of form:
     Function callbackProc(eventID: smallint;
     param: pointer;
     userData: pointer)}
): Bool stdcall;
```
PESetFieldMappingType

Use PESetFieldMappingType to set the field mapping type code for the specified report.

C Syntax

```c
BOOL CRPE_API PESetFieldMappingType
    short printJob,
    WORD mappingType );
```

Parameters

<table>
<thead>
<tr>
<th>printJob</th>
<th>Specifies the print job for which you want to set the field mapping type code.</th>
</tr>
</thead>
<tbody>
<tr>
<td>mappingType</td>
<td>The field mapping type code that you want to set. Use one of the PE_FM_XXX Field Mapping Type Constants (page 542).</td>
</tr>
</tbody>
</table>

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

```vb
Declare Function PESetFieldMappingType Lib "crpe32.d11" ( 
    ByVal printJob As Integer, ByVal mappingType As Integer ) As Integer
```

Delphi Syntax

```delphi
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

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to select a font.</td>
</tr>
<tr>
<td>sectionCode</td>
<td>Specifies the Section Codes (page 551) for the report section(s) for which you want to select a font. See Working with section codes, Page 22.</td>
</tr>
<tr>
<td>scopeCode</td>
<td>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.</td>
</tr>
<tr>
<td>faceName</td>
<td>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.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_FIELDS</td>
<td>Sets the default font for all field values in the report section specified.</td>
</tr>
<tr>
<td>PE_TEXT</td>
<td>Sets the default font for all text (that has not been entered as a field value) in the report section specified.</td>
</tr>
</tbody>
</table>
### fontFamily
Specifies the font family for the font you want to use. Use one of the following FF_XXX constants.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF_DONTCARE</td>
<td>No change.</td>
</tr>
<tr>
<td>FF_ROMAN</td>
<td>Variable pitch font with serifs.</td>
</tr>
<tr>
<td>FF_SWISS</td>
<td>Fixed pitch font without serifs.</td>
</tr>
<tr>
<td>FF_MODERN</td>
<td>Fixed-pitch font, with or without serifs.</td>
</tr>
<tr>
<td>FF_SCRIPT</td>
<td>Handwriting-like font.</td>
</tr>
<tr>
<td>FF_DECORATIVE</td>
<td>Fancy display font.</td>
</tr>
</tbody>
</table>

### 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.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT_PITCH</td>
<td>0x00</td>
</tr>
<tr>
<td>FIXED_PITCH</td>
<td>0x01</td>
</tr>
<tr>
<td>VARIABLE_PITCH</td>
<td>0x02</td>
</tr>
</tbody>
</table>

### 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.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSI_CHARSET</td>
<td>0</td>
</tr>
<tr>
<td>DEFAULT_CHARSET</td>
<td>1</td>
</tr>
<tr>
<td>SYMBOL_CHARSET</td>
<td>2</td>
</tr>
<tr>
<td>SHIFTJIS_CHARSET</td>
<td>128</td>
</tr>
<tr>
<td>HANGEUL_CHARSET</td>
<td>129</td>
</tr>
<tr>
<td>CHINESEBIG5_CHARSET</td>
<td>136</td>
</tr>
<tr>
<td>OEM_CHARSET</td>
<td>255</td>
</tr>
</tbody>
</table>

### 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.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FW_DONTCARE</td>
<td>0</td>
</tr>
<tr>
<td>FW_THIN</td>
<td>100</td>
</tr>
<tr>
<td>FW_EXTRALIGHT</td>
<td>200</td>
</tr>
<tr>
<td>FW_LIGHT</td>
<td>300</td>
</tr>
<tr>
<td>FW_NORMAL</td>
<td>400</td>
</tr>
<tr>
<td>FW_MEDIUM</td>
<td>500</td>
</tr>
<tr>
<td>FW_SEMIBOLD</td>
<td>600</td>
</tr>
<tr>
<td>FW_BOLD</td>
<td>700</td>
</tr>
<tr>
<td>FW_EXTRABOLD</td>
<td>800</td>
</tr>
<tr>
<td>FW_HEavy</td>
<td>900</td>
</tr>
<tr>
<td>FW_ULTRALIGHT</td>
<td>FW_EXTRALIGHT</td>
</tr>
<tr>
<td>FW_REGULAR</td>
<td>FW_NORMAL</td>
</tr>
<tr>
<td>FW_DEMIBOLD</td>
<td>FW_SEMIBOLD</td>
</tr>
<tr>
<td>FW_ULTRABOLD</td>
<td>FW_EXTRABOLD</td>
</tr>
<tr>
<td>FW_BLACK</td>
<td>FW_HEavy</td>
</tr>
</tbody>
</table>

**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 PESStartPrintJob, Page 437, or the results may be inconsistent or unexpected.
### VB Syntax

```vbnet
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

```delphi
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

```dbase
EXTERN CLOGICAL PESetFont (CWORD, CWORD, CWORD, CSTRING, 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 (PGetFormula, Page 291; PGetHandleString, Page 303; 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

```c
BOOL CRPE_API PESetFormula (  
    short printJob,  
    const char *formulaName,  
    const char FAR *formulaString );
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to set a new formula string.</td>
</tr>
<tr>
<td>formulaName</td>
<td>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.</td>
</tr>
<tr>
<td>formulaString</td>
<td>Specifies a pointer to the null-terminated string that you want to replace the existing formula string.</td>
</tr>
</tbody>
</table>

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 PESetPrintJob, Page 437 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

```c
BOOL CRPE_API PESetFormulaSyntax ( 
    short printJob, 
    PEFormulaSyntax FAR *formulaSyntax );
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to set formula syntax.</td>
</tr>
<tr>
<td>formulaSyntax</td>
<td>Specifies a pointer to <code>PEFormulaSyntax</code> (page 456), which will contain the information that you want to set.</td>
</tr>
</tbody>
</table>

Returns

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

Remarks

- If PESetFormulaSyntax is called before any Formula API is called, then the default `PE_FSTCRYSTAL` 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

```vb
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**

```c
BOOL CRPE_API PESetGraphAxisInfo (  
    short printJob,  
    short sectionN,  
    short graphN,  
    PEGraphAxisInfo FAR * graphAxisInfo );
```

**Parameters**

<table>
<thead>
<tr>
<th>printJob</th>
<th>Specifies the print job for which you want to set chart axis information.</th>
</tr>
</thead>
<tbody>
<tr>
<td>sectionN</td>
<td>Specifies the section of the report containing the chart for which you want to set chart axis information.</td>
</tr>
<tr>
<td>graphN</td>
<td>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.</td>
</tr>
<tr>
<td>graphAxisInfo</td>
<td>Specifies a pointer to <strong>PEGraphAxisInfo</strong> (page 458), which will contain the new information.</td>
</tr>
</tbody>
</table>

**Returns**

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

**VB Syntax**

```vb
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**

```delphi
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

```c
BOOL CRPE_API PESetGraphFontInfo ( 
    short printJob, 
    short sectionN, 
    short graphN, 
    WORD titleFontType, 
    PEFontColorInfo FAR *fontColourInfo );
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to set chart font information.</td>
</tr>
<tr>
<td>sectionN</td>
<td>Specifies the number of the section in which the chart appears. This parameter should be within the range obtained by <code>PEGetNSections</code>, Page 313.</td>
</tr>
<tr>
<td>graphN</td>
<td>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.</td>
</tr>
<tr>
<td>titleFontType</td>
<td>Uses one of the <code>PE_GTF_XXX</code> Graph Text Font Constants (page 546)</td>
</tr>
<tr>
<td>fontColourInfo</td>
<td>Specifies a pointer to <code>PEFontColorInfo</code>, which will contain the new information.</td>
</tr>
</tbody>
</table>

### Returns

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

### VB Syntax

```vb
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

```delphi
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**

```c
BOOL CRPE_API PESetGraphOptionInfo (
    short printJob,
    short sectionN,
    short graphN,
    PEGraphOptionInfo FAR *graphOptionInfo);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to set chart display information.</td>
</tr>
<tr>
<td>sectionN</td>
<td>Specifies the number of the section in which the chart appears. This parameter should be within the range obtained by <code>PEGetNSections</code>.</td>
</tr>
<tr>
<td>graphN</td>
<td>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.</td>
</tr>
<tr>
<td>graphOptionInfo</td>
<td>Specifies a pointer to <code>PEGraphOptionInfo</code> (page 461), which will contain the new information.</td>
</tr>
</tbody>
</table>

**Returns**

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

**VB Syntax**

```vb
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**

```delphi
function PESetGraphOptionInfo ( 
    printJob : Smallint; 
    sectionN : Smallint; 
    graphN : Smallint; 
    var graphOptionInfo : PEGraphOptionInfo): Bool; {$ifdef WIN32} stdcall; {$endif}
```

Seagate Crystal Report Engine
PESetGraphTextDefaultOption

Use PESetGraphTextDefaultInfo to enable or disable chart title defaults.

C Syntax

```c
BOOL CRPE_API PESetGraphTextDefaultOption (  
    short printJob,  
    short sectionN,  
    short graphN,  
    WORD titleType,  
    BOOL useDefault );
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to enable/disable chart title default option information.</td>
</tr>
<tr>
<td>sectionN</td>
<td>Specifies the 0-based index number of the section in which the chart appears. This parameter should be within the range obtained by PEGetNSections, Page 313.</td>
</tr>
<tr>
<td>graphN</td>
<td>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.</td>
</tr>
<tr>
<td>titleType</td>
<td>Specifies the title type. Use one of the PE_GTT_XXX Graph Title Type Constants (page 546).</td>
</tr>
<tr>
<td>useDefault</td>
<td>Specifies the Boolean value indicating whether or not chart title defaults are enabled.</td>
</tr>
</tbody>
</table>

Returns

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

VB Syntax

```vb
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**

```c
BOOL CRPE_API PESetGraphTextInfo (
    short printJob,
    short sectionN,
    short graphN,
    WORD titleType,
    LPCSTR title);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to set title text information.</td>
</tr>
<tr>
<td>sectionN</td>
<td>Specifies the 0-based index number of the section in which the chart appears.</td>
</tr>
<tr>
<td>graphN</td>
<td>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.</td>
</tr>
<tr>
<td>titleType</td>
<td>Specifies the title type. Use one of the <strong>PE_GTT_XXX Graph Title Type Constants</strong> (page 546).</td>
</tr>
<tr>
<td>title</td>
<td>Specifies a pointer to the string containing the title text.</td>
</tr>
</tbody>
</table>

**Returns**

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

**VB Syntax**

```vb
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**

```delphi
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

```c
BOOL CRPE_API PESetGraphTypeInfo (
    short printJob,
    short sectionN,
    short graphN,
    PEGraphTypeInfo FAR *graphTypeInfo );
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to set chart type information.</td>
</tr>
<tr>
<td>sectionN</td>
<td>Specifies the number of the section in which the chart appears. This parameter should be within the range obtained by <strong>PEGetNSections</strong>, Page 313.</td>
</tr>
<tr>
<td>graphN</td>
<td>Specifies 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.</td>
</tr>
<tr>
<td>graphTypeInfo</td>
<td>Specifies a pointer to <strong>PEGraphTypeInfo</strong> (page 464), which will contain the new information.</td>
</tr>
</tbody>
</table>

### Returns

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

### VB Syntax

```vb
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

```delphi
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

```c
BOOL CRPE_API PESetGroupCondition (  
    short printJob,  
    short sectionCode,  
    const char FAR *conditionField,  
    short condition,  
    short sortDirection );
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>printJob</code></td>
<td>Specifies the print job for which you want to change the group condition for a group section.</td>
</tr>
<tr>
<td><code>sectionCode</code></td>
<td>Specifies the code for the report section for which you want to set the group condition. See Working with section codes, Page 22.</td>
</tr>
<tr>
<td><code>conditionField</code></td>
<td>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, Page 303, with conditionFieldHandle and conditionFieldLength, returned by PEGetGroupCondition, Page 299.</td>
</tr>
<tr>
<td><code>condition</code></td>
<td>Specifies the condition that will trigger a summary. Use one of the PE_GC_XXX Group Condition Constants (page 547). Note that the constants available are different for different field types.</td>
</tr>
<tr>
<td><code>sortDirection</code></td>
<td>Use one of the PE_SF_XXX Sort Order Constants (page 551).</td>
</tr>
</tbody>
</table>

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 PESetStartPrintJob, Page 437 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you wish to set grouping options.</td>
</tr>
<tr>
<td>groupN</td>
<td>Specifies the 0-based group level number.</td>
</tr>
<tr>
<td>groupOptions</td>
<td>Specifies a pointer to PEGroupOptions (page 465).</td>
</tr>
</tbody>
</table>

Returns

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

- This function should be called before `PEStartPrintJob`, Page 437 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

```vbnet
Declare Function PESetGroupOptions Lib "crpe32.dll" (ByVal printJob As Integer, ByVal groupN As Integer, groupOptions As PEGroupOptions) As Integer
```

Delphi Syntax

```delphi
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`, Page 302; `PEGetHandleString`, Page 303; 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

```c
BOOL CRPE_API PESetGroupSelectionFormula (short printJob, const char FAR *formulaString);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>printJob</code></td>
<td>Specifies the print job for which you want to set a new group selection formula.</td>
</tr>
<tr>
<td><code>formulaString</code></td>
<td>Specifies a pointer to the null-terminated string you want to assign to the group selection formula.</td>
</tr>
</tbody>
</table>
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, Page 267.
● This function should be called before PEStartPrintJob, Page 437 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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to set new margins.</td>
</tr>
<tr>
<td>left</td>
<td>Specifies the left margin.</td>
</tr>
<tr>
<td>right</td>
<td>Specifies the right margin.</td>
</tr>
<tr>
<td>top</td>
<td>Specifies the top margin.</td>
</tr>
<tr>
<td>bottom</td>
<td>Specifies the bottom margin.</td>
</tr>
</tbody>
</table>

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

Remarks
- This function should be called before PESetPrintJob, Page 437, 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, Page 306.

C Syntax

```c
BOOL CRPE_API PESetNDetailCopies(
    short printJob,
    short nDetailCopies);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want set the number of copies to print.</td>
</tr>
<tr>
<td>nDetailCopies</td>
<td>Specifies the number of report copies you want to print.</td>
</tr>
</tbody>
</table>

Returns

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

Remarks

- This function should be called before PESetPrintJob, Page 437, or the results may be inconsistent or unexpected.
- To change top/bottom N group sorting order, use PESetGroupOptions, Page 402.

VB Syntax

```vb
Declare Function PESetNDetailCopies Lib "crpe32.dll" (
    ByVal printJob As Integer, ByVal nDetailCopies As Integer) As Integer
```

Delphi Syntax

```delphi
function PESetNDetailCopies(
    printJob: Word;
    nDetailCopies: smallint;
) : Bool stdcall;
```

dBASE for Windows Syntax

```dBASE
EXTERN CLOGICAL PESetNDetailCopies (CWORD, CWORD) CRPE.DLL
```
PESetNthGroupSortField

Use PESetNthGroupSortField to set one of the group sort fields in the specified report. This function can be used by itself to set a sort field and direction when there is not one already set, or to modify an existing sort field and direction when the sort field number, name, and direction are known.

The function can also be used as one of a series of functions (PEGetNGroupSortFields, Page 308, called once; PEGetNthGroupSortField, Page 319, or PEGetHandleString, Page 303, 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

```c
BOOL CRPE_API PESetNthGroupSortField(
    short printJob,
    short sortFieldN,
    const char FAR *name,
    short direction
);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to set a group sort field.</td>
</tr>
<tr>
<td>sortFieldN</td>
<td>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.</td>
</tr>
<tr>
<td>name</td>
<td>Specifies a pointer to the null-terminated string that contains the name of the group sort field.</td>
</tr>
<tr>
<td>direction</td>
<td>Specifies the sort directions. Use one of the PE_SF_XXX Sort Order Constants (page 551).</td>
</tr>
</tbody>
</table>

Returns

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

Remarks

This function should be called before PESetPrintJob, Page 437, or the results may be inconsistent or unexpected.
**Functions**

**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, Page 312, to retrieve the number of default values for the parameter field.

**C Syntax**

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to set a parameter default value.</td>
</tr>
<tr>
<td>parameterFieldName</td>
<td>Specifies a pointer to the string containing the parameter field name.</td>
</tr>
<tr>
<td>reportName</td>
<td>Specifies a pointer to the string containing the report name. See Remarks below.</td>
</tr>
<tr>
<td>index</td>
<td>Specifies the index number of the default value to be set.</td>
</tr>
<tr>
<td>valueInfo</td>
<td>Specifies a pointer to PEValueInfo (page 507), which will contain the default value.</td>
</tr>
</tbody>
</table>

**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**

```vbnet
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**

```delphi
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**

```c
BOOL CRPE_API PESetNthParameterField ( _
    short printJob,
    short parameterN,
    PEParameterFieldInfo FAR *parameterInfo );
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>printJob</code></td>
<td>Specifies the print job for which you want to set a parameter field value.</td>
</tr>
<tr>
<td><code>parameterN</code></td>
<td>Specifies the number of the parameter field in the report.</td>
</tr>
<tr>
<td><code>parameterInfo</code></td>
<td>Specifies a pointer to <code>PEParameterFieldInfo</code> (page 474) which is used to pass the parameter field value information. See Remarks below.</td>
</tr>
</tbody>
</table>

**Returns**

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

- For new development, the Default/CurrentValueSet members of PEPParameterFieldInfo must be set to FALSE and the following new calls used to access default and current value lists.
  - PEAddParameterCurrentRange, Page 262
  - PEAddParameterCurrentValue, Page 263
  - PEAddParameterDefaultValue, Page 264
  - PESetNthParameterDefaultValue, Page 408

- 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 PEPParameterFieldInfo (page 474).

- To determine if a parameter field is a stored procedure, use PEGetNthParameterType, Page 325 or PEGetNthParameterField, Page 324 functions.

- This function should be called before PESetPrintJob, Page 437 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 PEPParameterFieldInfo) As Integer

Delphi Syntax

function PESetNthParameterField (printJob: Word; varN: Smallint; var varInfo: PEPParameterFieldInfo): 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to set parameter value description information.</td>
</tr>
<tr>
<td>parameterFieldName</td>
<td>Specifies a pointer to the parameterFieldName for which you want to set the parameter value description.</td>
</tr>
<tr>
<td>reportName</td>
<td>Specifies a pointer to the report name. See Remarks below.</td>
</tr>
<tr>
<td>index</td>
<td>Specifies the index.</td>
</tr>
<tr>
<td>valueDesc</td>
<td>Specifies a pointer to the handle of the value description to be set.</td>
</tr>
</tbody>
</table>

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

```vbnet
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

```delphi
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, Page 315, called once; PEGetNthSortField, Page 327, or PEGetHandleString, Page 303, 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to set sort field information.</td>
</tr>
<tr>
<td>sortFieldN</td>
<td>Specifies the 0-based number of the sort fields that you want to set. The</td>
</tr>
<tr>
<td></td>
<td>first sort field is field 0. If the report has N sort fields, the function</td>
</tr>
<tr>
<td></td>
<td>can be called with sortFieldN between 0 and N-1 to replace an existing sort</td>
</tr>
<tr>
<td></td>
<td>field. If the report has N sort fields, you can call the function with</td>
</tr>
<tr>
<td></td>
<td>sortFieldN = N to add a new sort field to the end of the list of existing</td>
</tr>
<tr>
<td></td>
<td>sort fields. If N=0, the function will add the first sort field.</td>
</tr>
<tr>
<td>name</td>
<td>Specifies a pointer to the null-terminated string that contains the name of</td>
</tr>
<tr>
<td></td>
<td>the sort field.</td>
</tr>
<tr>
<td>direction</td>
<td>Specifies the sort direction. Use one of the PE_SF_XXX Sort Order Constants</td>
</tr>
<tr>
<td></td>
<td>(page 551).</td>
</tr>
</tbody>
</table>

Returns

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

Remarks

This function should be called before PESetPrintJob, Page 437 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

```c
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, Page 331 to identify the location of a table and then to change it.

C Syntax

```c
BOOL CRPE_API PESetNthTableLocation (  
  short printJob,  
  short tableN,  
  PETableLocation FAR *location );
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the handle of the print job for which you want to set a table’s location.</td>
</tr>
<tr>
<td>tableN</td>
<td>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.</td>
</tr>
<tr>
<td>location</td>
<td>Specifies the pointer to PETableLocation (page 501). The format of the string will be depend on the type of database specified.</td>
</tr>
</tbody>
</table>

Returns

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

VB Syntax

```vbnet
Declare Function PESetNthTableLocation Lib "crpe32.dll" ( ByVal printJob As Integer, ByVal TableN As Integer, Location As PETableLocation ) As Integer
```
Delphi Syntax

function PESetNthTableLogOnInfo(
    printJob: Word;
    tableN: smallint;
    var logOnInfo: PELogOnInfo;
    var propagateAcrossTables: Boolean;
): Boolean;

C Syntax

BOOL CRPE_API PESetNthTableLogOnInfo(
    short printJob,
    short tableN,
    PELogOnInfo FAR *logOnInfo,
    BOOL propagateAcrossTables );

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to set table log on information.</td>
</tr>
<tr>
<td>tableN</td>
<td>Specifies the 0-based number of the table for which you want to set log on</td>
</tr>
<tr>
<td></td>
<td>information. The first table is table 0. The last table is N-1.</td>
</tr>
<tr>
<td>logOnInfo</td>
<td>Specifies a pointer to the PELogOnInfo (page 470).</td>
</tr>
<tr>
<td>propagateAcrossTables</td>
<td>If set to TRUE, the program will apply the new log on information to any</td>
</tr>
<tr>
<td></td>
<td>other tables in the report that had the same original server and database</td>
</tr>
<tr>
<td></td>
<td>names as the specified table. If set to FALSE, the program will apply the</td>
</tr>
<tr>
<td></td>
<td>new log on information only to the table specified.</td>
</tr>
</tbody>
</table>

Returns

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

Remarks

- See PELogOnServer, Page 359 for additional comments regarding PELogOnInfo (page 470).
- 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** (page 470).

**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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to change the MS Access session information.</td>
</tr>
<tr>
<td>tableN</td>
<td>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.</td>
</tr>
<tr>
<td>privateInfo</td>
<td>Specifies a pointer to <strong>PETablePrivateInfo</strong> (page 502).</td>
</tr>
</tbody>
</table>

**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}

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to change the MS Access session</td>
</tr>
<tr>
<td></td>
<td>information.</td>
</tr>
<tr>
<td>tableN</td>
<td>Specifies the 0-based number of the table for which you want to open the</td>
</tr>
<tr>
<td></td>
<td>session. The first table is table 0. The last table is N-1.</td>
</tr>
<tr>
<td>sessionInfo</td>
<td>Specifies a pointer to PESessionInfo (page 493).</td>
</tr>
<tr>
<td>propagateAcrossTables</td>
<td>Boolean value indicating whether the session information should be used for</td>
</tr>
<tr>
<td></td>
<td>opening all tables being used in the report.</td>
</tr>
</tbody>
</table>

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, Page 416.
● 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 (page 470) and calling PESetNthTableLogOnInfo, Page 414.

**VB Syntax**

```vbnet
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**

```delphi
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**

```c
BOOL CRPE_API PESetParameterMinMaxValue(
    short printJob,
    const char FAR *parameterFieldName,
    const char FAR *reportName,
    PEValueInfo FAR *valueMin,
    PEValueInfo FAR *valueMax );
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to set minimum and/or maximum parameters values.</td>
</tr>
<tr>
<td>parameterFieldName</td>
<td>Specifies a pointer to the string containing the parameter name.</td>
</tr>
<tr>
<td>reportName</td>
<td>Specifies a pointer to the string containing the report name. See Remarks below.</td>
</tr>
</tbody>
</table>
valueMin | Specifies a pointer to **PEValueInfo** (page 507), which contains minimum value information. See Remarks below.
---|---
valueMax | Specifies a pointer to **PEValueInfo** (page 507), 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**

```vbnet
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**

```delphi
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 (page 478).

C Syntax

```c
BOOL CRPE_API PESetParameterPickListOption ( 
    short printJob, 
    const char FAR *parameterFieldName, 
    const char FAR *reportName, 
    PEParameterPickListOption FAR *pickListOption );
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to set the parameter pick list options.</td>
</tr>
<tr>
<td>parameterFieldName</td>
<td>Specifies a pointer to the parameterFieldName for which you want to set pick list options.</td>
</tr>
<tr>
<td>reportName</td>
<td>Specifies a pointer to the report name. See Remarks below.</td>
</tr>
<tr>
<td>pickListOption</td>
<td>Specifies a pointer to PEParameterPickListOption (page 478), which will contain the new information.</td>
</tr>
</tbody>
</table>

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

```vb
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

```delphi
function PESetParameterPickListOption (  
  printJob           : Smallint;  
  parameterFieldName : PChar;  
  reportName         : PChar;  
  var pickListOption : PEPParameterPickListOption): 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**

```c
BOOL CRPE_API PESetParameterValueInfo(  
  short printJob,  
  const char FAR *parameterFieldName,  
  const char FAR *reportName,  
  PEPParameterValueInfo FAR *valueInfo );
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to set parameter value information.</td>
</tr>
<tr>
<td>parameterFieldName</td>
<td>Specifies a pointer to the string containing the parameter field name.</td>
</tr>
<tr>
<td>reportName</td>
<td>Specifies a pointer to the string containing the report name. See Remarks below.</td>
</tr>
<tr>
<td>valueInfo</td>
<td>Specifies a pointer to PEPParameterValueInfo (page 479), which contains the parameter value information.</td>
</tr>
</tbody>
</table>

**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**

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

<table>
<thead>
<tr>
<th>printJob</th>
<th>Specifies the print job for which you want to set the print date.</th>
</tr>
</thead>
<tbody>
<tr>
<td>year</td>
<td>Specifies the year component of the print date. Enter a 4 digit year value (1994, 1993, etc.).</td>
</tr>
<tr>
<td>month</td>
<td>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.</td>
</tr>
<tr>
<td>day</td>
<td>Specifies the day component of the print date. Enter the actual day of the month you want to use (7, 18, 28, etc.).</td>
</tr>
</tbody>
</table>

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 PESetPrintJob, Page 437, 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 (page 480). 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to set printing options.</td>
</tr>
<tr>
<td>options</td>
<td>Specifies a pointer to PEPrintOptions (page 480). 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, Page 373.</td>
</tr>
</tbody>
</table>

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 PEReportOptions (page 485). Use this function any time you want to set any of the report options found in the Report|Options dialog box in the Seagate Crystal Reports Designer.

C Syntax

BOOL CRPE_API PESetReportOptions (short printJob,
PEReportOptions FAR *reportOptions );

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to set report options.</td>
</tr>
<tr>
<td>reportOptions</td>
<td>Specifies a pointer to PEReportOptions (page 485).</td>
</tr>
</tbody>
</table>

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 Seagate Crystal Reports.

C Syntax

```c
BOOL CRPE_API PESetReportSummaryInfo (  
    short printJob,  
    PEReportSummaryInfo FAR *summaryInfo );
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to set summary information.</td>
</tr>
<tr>
<td>summaryInfo</td>
<td>Specifies the pointer to PEROportSummaryInfo (page 488).</td>
</tr>
</tbody>
</table>

Returns

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

Remarks

This function should be called before PESetReportSummaryInfo, Page 437, or the results may be inconsistent or unexpected.

VB Syntax

```vb
Declare Function PESetReportSummaryInfo Lib "crpe32.dll" (  
    ByVal printJob as Integer, summaryInfo as PEROportSummaryInfo ) as Integer
```

Delphi Syntax

```delphi
function PESetReportSummaryInfo (  
    printJob: Word;  
    var summaryInfo: PEROportSummaryInfo  
): 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: PESetReportTitle, Page 342; PESetHandleString, Page 303; 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

```c
BOOL CRPE_API PESetReportTitle (  
    short printJob,  
    const char FAR *title );
```

Parameters

<table>
<thead>
<tr>
<th>printJob</th>
<th>Specifies the print job for which you want to set the report title.</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>Specifies a pointer to the null-terminated string containing the new title that you want to assign to the report.</td>
</tr>
</tbody>
</table>

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`, Page 437, or the results may be inconsistent or unexpected.

VB Syntax

```vb
Declare Function PESetReportTitle Lib "crpe32.dll" (  
    ByVal printJob As Integer, ByVal Title As String ) As Integer
```

Delphi Syntax

```delphi
function PESetReportTitle (  
    printJob: Word;  
    title: PChar  
): Bool stdcall;
```

dBASE for Windows Syntax

```dBASE
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** (page 491). 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

```c
BOOL CRPE_API PESetSectionFormat (  
    short printJob,  
    short sectionCode,  
    PESectionOptions FAR *options );
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to set section formatting options.</td>
</tr>
<tr>
<td>sectionCode</td>
<td>Specifies the Section Codes (page 551) for the report section(s) for which you want to set formatting options. See <strong>Working with section codes, Page 22</strong>.</td>
</tr>
<tr>
<td>options</td>
<td>Specifies a pointer to <strong>PESectionOptions</strong> (page 491). Use this structure to set your section options.</td>
</tr>
</tbody>
</table>

### 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, Page 437** or the results may be inconsistent or unexpected.

### VB Syntax

```vbnet
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to set a new selection formula string.</td>
</tr>
<tr>
<td>sectionCode</td>
<td>Specifies the Section Codes (page 551), for the report section(s) for which you want to set formatting options. See Working with section codes, Page 22.</td>
</tr>
<tr>
<td>formulaName</td>
<td>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 (page 533).</td>
</tr>
<tr>
<td>formulaString</td>
<td>Specifies a pointer to the null-terminated string that you want to assign to the format formula.</td>
</tr>
</tbody>
</table>

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 PESetPrintJob, Page 437, 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to set section height information.</td>
</tr>
<tr>
<td>sectionCode</td>
<td>Specifies the Section Codes (page 551) for the section(s) for which you want to set section height information. See Working with section codes, Page 22.</td>
</tr>
<tr>
<td>height</td>
<td>The section height measured in twips.</td>
</tr>
</tbody>
</table>

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, Page 349; PEGetHandleString, Page 303; 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

<table>
<thead>
<tr>
<th>printJob</th>
<th>Specifies the print job for which you want to set a new selection formula string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>formulaString</td>
<td>Specifies a pointer to the null-terminated string that you want to assign to the selection formula.</td>
</tr>
</tbody>
</table>

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, Page 269.
- This function should be called before PESetPrintJob, Page 437 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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to set an SQL expression.</td>
</tr>
<tr>
<td>expressionName</td>
<td>Specifies a pointer to the string containing the expression name.</td>
</tr>
<tr>
<td>expressionString</td>
<td>Specifies a pointer to the string containing the SQL expression.</td>
</tr>
</tbody>
</table>

**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

```delphi
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

```c
BOOL CRPE_API PESetSQLQuery (  
    short printJob,  
    const char FAR *queryString );
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to modify the SQL query.</td>
</tr>
<tr>
<td>queryString</td>
<td>Specifies a pointer to the null-terminated string that you want to use to replace the existing SQL query.</td>
</tr>
</tbody>
</table>

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 Seagate 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, Page 429, and let Seagate 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, Page 368), and if events for the preview window have been enabled (see PEEnableEvent, Page 281).

C Syntax
BOOL CRPE_API PESetTrackCursorInfo ( short printJob, PETrackCursorInfo FAR *cursorInfo );

Parameters
<table>
<thead>
<tr>
<th>printJob</th>
<th>Specifies the print job for which you want to track the mouse cursor.</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursorInfo</td>
<td>Specifies a pointer to PETrackCursorInfo (page 504).</td>
</tr>
</tbody>
</table>

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, Page 368.

C Syntax

BOOL CRPE_API PESetWindowOptions (
    short printJob,
    PEWindowOptions FAR *options );

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to set preview window display options.</td>
</tr>
<tr>
<td>options</td>
<td>Specifies a pointer to a PEWindowOptions (page 509).</td>
</tr>
</tbody>
</table>

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, Page 309, to determine the number of pages available in the specified report when using PEShowNthPage, for example.

C Syntax

```c
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)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to indicate the page to be displayed.</td>
</tr>
</tbody>
</table>

- The following parameter applies only to PEShowNthPage.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pageN</td>
<td>Specifies the 1-based number indicating the report page that you want to display when calling PEShowNthPage. See Remarks below.</td>
</tr>
</tbody>
</table>

Returns

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

Remarks

Using PEGetJobStatus, Page 304, 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**

```vbnet
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**

```delphi
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**

```sql
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

```c
BOOL CRPE_API PEShowPrintControls (
    short printJob,
    BOOL showPrintControls );
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to toggle the display of print controls.</td>
</tr>
<tr>
<td>showPrintControls</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

```vb
Declare Function PEShowPrintControls Lib "crpe32.d11" (  
    ByVal printJob As Integer, ByVal ShowPrintControls As Integer ) As Integer
```

Delphi Syntax

```delphi
function PEShowPrintControls (  
    printJob: Word;  
    showPrintControls: Bool  
): Bool stdcall;
```

dBASE for Windows Syntax

```dBASE
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

```c
BOOL CRPE_API PEStartPrintJob ( 
    short printJob, 
    BOOL waitUntilDone );
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job you want to start.</td>
</tr>
<tr>
<td>waitUntilDone</td>
<td>BOOL. Reserved. This parameter must always be set to TRUE.</td>
</tr>
</tbody>
</table>

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, Page 272.

VB Syntax

```vb
Declare Function PEStartPrintJob Lib “crpe32.dll” (ByVal printJob _ 
    As Integer, ByVal WaitOrNot As Integer) As Integer
```

Delphi Syntax

```delphi
function PEStartPrintJob ( 
    printJob: Word; 
    waitUntilDone: Bool 
): Bool stdcall;
```

dBASE for Windows Syntax

```dBASE
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**

```c
BOOL CRPE_API PETestNthTableConnectivity ( 
    short printJob, 
    short tableN );
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to test a table’s connection settings.</td>
</tr>
<tr>
<td>tableN</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Connection Failure</th>
<th>Error Code</th>
<th>Update with Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unable to begin a session.</td>
<td>PE_ERR_DATABASESESSION</td>
<td>PESetNthTableSessionInfo, Page 416</td>
</tr>
<tr>
<td>Unable to log on to a server.</td>
<td>PE_ERR_DATABASELOGON</td>
<td>PESetNthTableLogOnInfo, Page 414</td>
</tr>
<tr>
<td>Unable to open the table.</td>
<td>PE_ERR_DATABASELOCATION</td>
<td>PESetNthTableLocation, Page 413</td>
</tr>
</tbody>
</table>

- 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**

```c
BOOL CRPE_API PEZoomPreviewWindow ( 
    short printJob, 
    short level );
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printJob</td>
<td>Specifies the print job for which you want to set the magnification level in the preview window.</td>
</tr>
<tr>
<td>level</td>
<td>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 <strong>Zoom Level Constants</strong> (page 553).</td>
</tr>
</tbody>
</table>

**Returns**

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

**VB Syntax**

```vbnet
Declare Function PEZoomPreviewWindow Lib “crpe32.dll” ( 
    ByVal printJob As Integer, ByVal ZoomLevel As Integer ) As Integer
```

**Delphi Syntax**

```delphi
function PEZoomPreviewWindow ( 
    printJob: Word; 
    level: smallint 
): Bool stdcall;
```

**dBASE for Windows Syntax**

```dBASE
EXTERN CLOGICAL PEZoomPreviewWindow (CWORD, CWORD)CRPE.DLL
```
PRINT ENGINE STRUCTURES

This section includes the following CRPE structures, listed alphabetically.

- PECloseButtonClickedEventInfo, Page 443
- PEDrillOnDetailEventInfo, Page 443
- PEDrillOnGroupEventInfo, Page 445
- PEEnableEventInfo, Page 446
- PEEExportOptions, Page 448
- PEFIELDMappingEventInfo, Page 452
- PEFIELDValueInfo, Page 453
- PEFONToColorInfo, Page 454
- PEFORMulasSyntax, Page 456
- PEGENERalPrintWindowEventInfo, Page 456
- PEGraphAxisInfo, Page 458
- PEGraphOptionInfo, Page 461
- PEGraphTypeInfo, Page 464
- PEGroupOptions, Page 465
- PEGroupTreeButtonClickedEventInfo, Page 467
- PEHyperlinkEventInfo, Page 468
- PEJobInfo, Page 468
- PELaunchSeagateAnalysisEventInfo, Page 470
- PELogOnInfo, Page 470
- PMouseClickEventInfo, Page 472
- POBJectInfo, Page 473
- POleObjectInfo, Page 474
- PEPParameterFieldInfo, Page 474
- PEParameterPickListOption, Page 478
- PEPParameterValueInfo, Page 479
- PEPrintOptions, Page 480
- PEReadingRecordsEventInfo, Page 482
PEReportFieldMappingInfo, Page 483
PEReportOptions, Page 485
PEReportSummaryInfo, Page 488
PESearchButtonClickedEventInfo, Page 490
PESectionOptions, Page 491
PESessionInfo, Page 493
PEShowGroupEventInfo, Page 495
PEStartEventInfo, Page 496
PEStopEventInfo, Page 497
PESubreportInfo, Page 498
PETableDifferenceInfo, Page 499
PETableLocation, Page 501
PETablePrivateInfo, Page 502
PETableType, Page 503
PETrackCursorInfo, Page 504
PEValueInfo, Page 507
PEWindowOptions, Page 509
PEZoomLevelChangingEventInfo, Page 511
UXDDiskOptions, Page 512
UXDMAPIOptions, Page 513
UXDSMIOptions, Page 514
UXDPostFolderOptions, Page 515
UXDVIMOptions, Page 516
UXFCharSeparatedOptions, Page 517
UXFCommaTabSeparatedOptions, Page 518
UXFDIFOptions, Page 519
UXFHTML3Options, Page 520
UXFODBCOptions, Page 520
UXFPaginatedTextOptions, Page 521
UXFRecordStyleOptions, Page 522
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

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StructSize</td>
<td>Specifies the size of the PECloseButtonClickedEventInfo structure. Initialize this number to PE_SIZEOF_CLOSE_BUTTON_CLICKED_EVENT_INFO.</td>
</tr>
<tr>
<td>viewIndex</td>
<td>Specifies the 0-based index number indicating which view is going to be closed.</td>
</tr>
<tr>
<td>windowHandle</td>
<td>Specifies the handle for the frame window on which the button is placed.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StructSize</td>
<td>Specifies the size of the PEDrillOnDetailEventInfo structure. Initialize this number to PE_SIZEOF_DRILL_ON_DETAIL_EVENT_INFO.</td>
</tr>
<tr>
<td>selectedFieldIndex</td>
<td>The 0-based index indicating which drill-down field was selected. Contains -1 if no field was selected.</td>
</tr>
<tr>
<td>windowHandle</td>
<td>Frame window handle where the drill on detail event happens.</td>
</tr>
<tr>
<td>fieldValueList</td>
<td>Points to an array of PEFieldValue. Memory pointed by fieldValueList is freed after calling the callback function.</td>
</tr>
<tr>
<td>nFieldValue</td>
<td>The 1-based index of the value in field value list (for example, if the value is listed second, the number would be 2).</td>
</tr>
</tbody>
</table>

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**

```c
typedef struct PEDrillOnGroupEventInfo {
    WORD StructSize;
    WORD drillType;
    long windowHandle;
    char **groupList;
    WORD groupLevel;
} PEDrillOnGroupEventInfo;
```

**Members**

<table>
<thead>
<tr>
<th><strong>StructSize</strong></th>
<th>Specifies the size of the PEDrillOnGroupEventInfo structure. Initialize this number to PE_SIZEOF_DRILL_ON_GROUP_EVENT_INFO.</th>
</tr>
</thead>
<tbody>
<tr>
<td>drillType</td>
<td>Specifies the type of drill down that is used. Use one of the following values.</td>
</tr>
<tr>
<td><strong>Constant</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>PE_DE_ON_GROUP</td>
<td>Drill-down on a group summary or subtotal.</td>
</tr>
<tr>
<td>PE_DE_ON_GROUPTREE</td>
<td>Drill-down a Group Tree node.</td>
</tr>
<tr>
<td>PE_DE_ON_GRAPH</td>
<td>Drill-down a group graph object.</td>
</tr>
<tr>
<td>PE_DE_ON_MAP</td>
<td>Drill-down on a map region.</td>
</tr>
<tr>
<td>PE_DE_ON_SUBREPORT</td>
<td>Drill-down on a subreport.</td>
</tr>
</tbody>
</table>

windowHandle Frame window handle where the event happens.

groupList 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.

groupLevel The number of the group name in the group list.

**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, Page 281, 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 mouseClickedEvent;
    short hyperlinkEvent;
    short launchSeagateAnalysisEvent;
} PEEnableEventInfo;
Members
Each member of type short can be set to TRUE, FALSE, or PE_UNCHANGED for no change.

<table>
<thead>
<tr>
<th>StructSize</th>
<th>Specifies the size of the PEEnableEventInfo structure. Initialize this number to PE_SIZEOF_ENABLE_EVENT_INFO.</th>
</tr>
</thead>
<tbody>
<tr>
<td>startStopEvent</td>
<td>Boolean value, or PE_UNCHANGED for no change. Start/Stop event.</td>
</tr>
<tr>
<td>readingRecordEvent</td>
<td>Boolean value, or PE_UNCHANGED for no change. Reading record event.</td>
</tr>
<tr>
<td>printWindowButtonEvent</td>
<td>Boolean value, or PE_UNCHANGED for no change. Print window button event.</td>
</tr>
<tr>
<td>drillEvent</td>
<td>Boolean value, or PE_UNCHANGED for no change. Drill event.</td>
</tr>
<tr>
<td>closePrintWindowEvent</td>
<td>Boolean value, or PE_UNCHANGED for no change. Close print window event.</td>
</tr>
<tr>
<td>activatePrintWindowEvent</td>
<td>Boolean value, or PE_UNCHANGED for no change. Activate print window event.</td>
</tr>
<tr>
<td>fieldMappingEvent</td>
<td>Boolean value, or PE_UNCHANGED for no change. Field mapping event.</td>
</tr>
<tr>
<td>mouseClickEvent</td>
<td>Boolean value, or PE_UNCHANGED for no change. Mouse click event.</td>
</tr>
<tr>
<td>hyperlinkEvent</td>
<td>Boolean value, or PE_UNCHANGED for no change. Hyperlink event.</td>
</tr>
<tr>
<td>launchSeagateAnalysisEvent</td>
<td>Boolean value, or PE_UNCHANGED for no change. Launch Seagate Analysis event.</td>
</tr>
</tbody>
</table>

Remarks
- By default, all events are disabled. Use PEEnableEvent, Page 281, 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

```delphi
type
    PEEnableEventInfo = record
        StructSize: Word;
        startStopEvent: smallint;
        readingRecordEvent: smallint;
        printWindowButtonEvent: smallint;
        drillEvent: smallint;
        closePrintWindowEvent: smallint;
        activatePrintWindowEvent: smallint;
        fieldMappingEvent: smallint;
        mouseClickedEvent: smallint;
    end;
```

PEExportOptions

PEExportOptions contains file format and output destination information that is retrieved by `PEGetExportOptions, Page 289`, and used `PEExportTo, Page 283`, when exporting reports.

C Syntax

```c
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.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StructSize</td>
<td>Specifies the size of the PEExportOptions structure. Initialize the member to PE_SIZEOF_EXPORT_OPTIONS.</td>
</tr>
<tr>
<td>formatDLLName</td>
<td>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.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>To export in this format</th>
<th>Use this DLL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seagate Crystal Reports Format</td>
<td>u2fcr.dll</td>
</tr>
<tr>
<td>Data Interchange Format</td>
<td>u2fwordw.dll</td>
</tr>
</tbody>
</table>
**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.

<table>
<thead>
<tr>
<th>To export a report in this format</th>
<th>Use this formatType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seagate Crystal Reports Format</td>
<td>UXFCrystalReportType</td>
</tr>
<tr>
<td>Data Interchange Format</td>
<td>UXFDIFType</td>
</tr>
<tr>
<td>Word for Windows Format</td>
<td>UXFWordWinType</td>
</tr>
<tr>
<td>Word for DOS Format</td>
<td>UXFWordDosType</td>
</tr>
<tr>
<td>WordPerfect Format</td>
<td>UXFWordPerfectType</td>
</tr>
<tr>
<td>Quattro Pro 5.0 (WB1) Format</td>
<td>UXFQP5Type</td>
</tr>
<tr>
<td>Record Style Format (column of values)</td>
<td>UXFRecordType</td>
</tr>
<tr>
<td>Rich Text Format</td>
<td>UXFRichTextFormatType</td>
</tr>
<tr>
<td>Comma Separated Values Format (CSV)</td>
<td>UXFCommaSeparatedType</td>
</tr>
<tr>
<td>Tab Separated Values Format</td>
<td>UXFTabSeparatedType</td>
</tr>
<tr>
<td>Character Separated Values Format</td>
<td>UXFCharSeparatedType</td>
</tr>
<tr>
<td>Text Format (ASCII)</td>
<td>UXFTextType</td>
</tr>
<tr>
<td>Paginated Text Format (ASCII)</td>
<td>UXFPaginatedTextType</td>
</tr>
<tr>
<td>Tab Separated Text Format</td>
<td>UXFTabbedTextType</td>
</tr>
<tr>
<td>Lotus 1-2-3 (WK3)</td>
<td>UXFLotusWk3Type</td>
</tr>
<tr>
<td>Excel 4.0</td>
<td>UXFXls4Type</td>
</tr>
<tr>
<td>Excel 5.0</td>
<td>UXFXls5Type</td>
</tr>
<tr>
<td>Excel 5.0 Tabular</td>
<td>UXFXlsTypeTab</td>
</tr>
<tr>
<td>ODBC</td>
<td>UXFODBCType</td>
</tr>
<tr>
<td>HTML</td>
<td>UXFHTML3Type</td>
</tr>
<tr>
<td>Microsoft Internet Explorer 2 HTML</td>
<td>UXFExplorer2Type</td>
</tr>
<tr>
<td>Netscape 2 HTML</td>
<td>UXFNetcape2Type</td>
</tr>
</tbody>
</table>

**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 PESetStartPrintJob, Page 437, is called.

<table>
<thead>
<tr>
<th>To export a report in this format</th>
<th>Use this structure if you want to hard code formatOptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Interchange Format</td>
<td>UXFDIFOptions, Page 519</td>
</tr>
<tr>
<td>Record Style Format (column of values)</td>
<td>UXFRecordStyleOptions, Page 522</td>
</tr>
</tbody>
</table>
### Comma Separated Values (CSV)
- **Function**: UXFCommaTabSeparatedOptions, Page 518

### TabSeparated Values
- **Function**: UXFCommaTabSeparatedOptions, Page 518

### Character Separated Values
- **Function**: UXFCharSeparatedOptions, Page 517

### Paginated Text
- **Function**: UxFPaginatedTextOptions, Page 521

### Excel (Tabular)
- **Function**: UXFODBCOptions, Page 520

### HTML Format
- **Function**: UXFHTML3Options, Page 520

<table>
<thead>
<tr>
<th>destinationDLL Name</th>
<th>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.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>To export a report to this destination</th>
<th>Use this DLL name</th>
<th>Use this DLL name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk File</td>
<td>uxddisk.dll</td>
<td>u2ddisk.dll</td>
</tr>
<tr>
<td>E-Mail (MAPI)</td>
<td>uxdmapi.dll</td>
<td>u2dmapi.dll</td>
</tr>
<tr>
<td>E-Mail (VIM)</td>
<td>uxdvim.dll</td>
<td>u2dvim.dll</td>
</tr>
<tr>
<td>Microsoft Exchange</td>
<td>uxdpost.dll</td>
<td>u2dpost.dll</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>destinationType</th>
<th>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.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>To export a report to this destination</th>
<th>Use this destinationType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk File</td>
<td>UXDDiskType</td>
</tr>
<tr>
<td>E-Mail (MAPI)</td>
<td>UXMAPIType</td>
</tr>
<tr>
<td>E-Mail (VIM)</td>
<td>UXDVIMType</td>
</tr>
<tr>
<td>Microsoft Exchange</td>
<td>UXDExchFolderType</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>destinationOptions</th>
<th>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, Page 437, is called.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>To export a report to this destination</th>
<th>Use this structure if you want to hard code destinationOptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk File</td>
<td>UXDDiskOptions, Page 512</td>
</tr>
<tr>
<td>E-Mail (MAPI)</td>
<td>UXMAPIOptions, Page 513</td>
</tr>
</tbody>
</table>
Remarks
Note that both the formatOptions and destinationOptions members must be pointing to a valid address until PEStartPrintJob, Page 437, is called.

VB Type Listing

Type PEExportOptions
StructSize As Integer
FormatDLLName As String * PE_DLL_NAME_LEN
FormatTypeID As Integer
FormatTypeID2 As Integer
FormatOptions1 As Integer
FormatOptions2 As Integer
DestinationDLLName As String * PE_DLL_NAME_LEN
DestinationTypeID As Integer
DestinationTypeID2 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;
formatTypeID: dWord;
formatOptions: Pointer;
destinationDLLName: PEDllNameType;
destinationTypeID: 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

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StructSize</td>
<td>Specifies the size of the PEFieldMappingEventInfo structure. Initialize the member to PE_SIZEOF_FIELDMAPPING_EVENT_INFO.</td>
</tr>
<tr>
<td>reportFields</td>
<td>A pointer to an array of pointers to PEReportFieldMappingInfo (page 483), containing information about fields in the report.</td>
</tr>
<tr>
<td>nReportFields</td>
<td>Size of the reportFields array (equivalent to the number of fields in the report).</td>
</tr>
<tr>
<td>databaseFields</td>
<td>A pointer to an array of pointers to PEReportFieldMappingInfo (page 483) data members containing information about fields in the new database file.</td>
</tr>
<tr>
<td>nDatabaseFields</td>
<td>Size of the databaseFields array.</td>
</tr>
</tbody>
</table>

Remarks

To map a report field to a database field the member mappingTo of each PEReportFieldMappingInfo (page 483), 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;

    PEFieldMappingEventInfo = record
        StructSize : Word;
        reportFields : PEFieldMappingInfoDoublePtr;
        nReportFields : Word;
        databaseFields : PEFieldMappingInfoDoublePtr;
        nDatabaseFields : Word;
    end;
PEFieldValueInfo
Specifies a field value in the fieldValueList of PEDrillOnDetailEventInfo, Page 443.

C Syntax

```c
typedef struct PEFieldValueInfo {
    WORD StructSize;
    WORD ignored;
    char fieldName [PE_FIELD_NAME_LEN];
    PEValueInfo fieldValue;
} PEFieldValueInfo;
```

<table>
<thead>
<tr>
<th>Members</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>StructSize</td>
<td>Specifies the size of the PEFieldValueInfo structure. Initialize the member to PE_SIZEOF_FIELD_VALUE_INFO.</td>
</tr>
<tr>
<td>ignored</td>
<td>For 4 byte alignment. Ignore.</td>
</tr>
<tr>
<td>fieldName</td>
<td>Specifies the formula form of a field name (of length PE_FIELD_NAME_LEN = 512).</td>
</tr>
<tr>
<td>fieldValue</td>
<td>Specifies the field value for the selected Details area for the field specified by the field name.</td>
</tr>
</tbody>
</table>

Remarks
A selected Details area corresponds to a database record.

VB Type Listing

Type PEFieldValueInfo
```vbnet
    StructSize As Integer
    ignored As Integer
    fieldName As String
    fieldValue As PEValueInfo
End Type
```

Delphi Record Listing

```delphi
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

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StructSize</td>
<td>Specifies the size of the PEFontColorInfo structure. Initialize this member to PE_SIZEOF_FONT_COLOR_INFO.</td>
</tr>
<tr>
<td>faceName</td>
<td>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 (&quot;&quot;) for no change.</td>
</tr>
<tr>
<td>fontFamily</td>
<td>Specifies the font family for the font. Use one of the following FF_XXX Font Family Constants.</td>
</tr>
<tr>
<td>Constant</td>
<td>Description</td>
</tr>
<tr>
<td>FF_DONTCARE</td>
<td>No change.</td>
</tr>
<tr>
<td>FF_ROMAN</td>
<td>Variable pitch font with serifs.</td>
</tr>
<tr>
<td>FF_SWISS</td>
<td>Fixed pitch font without serifs.</td>
</tr>
<tr>
<td>FF_MODERN</td>
<td>Fixed-pitch font, with or without serifs.</td>
</tr>
<tr>
<td>FF_SCRIPT</td>
<td>Handwriting-like font.</td>
</tr>
<tr>
<td>FF_DECORATIVE</td>
<td>Fancy display font.</td>
</tr>
<tr>
<td>fontPitch</td>
<td>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.</td>
</tr>
<tr>
<td>charSet</td>
<td>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.</td>
</tr>
<tr>
<td>Member</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>pointSize</td>
<td>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.</td>
</tr>
<tr>
<td>isItalic</td>
<td>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.</td>
</tr>
<tr>
<td>isUnderlined</td>
<td>Specifies whether the font selected should be underlined. Use TRUE for Underline, FALSE for no Underline, or PE_UNCHANGED for the current default setting.</td>
</tr>
<tr>
<td>isStruckOut</td>
<td>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.</td>
</tr>
<tr>
<td>weight</td>
<td>Specifies the weight of the font. Use a constant value from the weight values defined in WINDOWS.H. Use 0 for no change.</td>
</tr>
<tr>
<td>color</td>
<td>Specifies the RGB color value contained in COLORREF (page 523). Use PE_UNCHANGED_COLOR for the current default setting.</td>
</tr>
<tr>
<td>twipSize</td>
<td>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.</td>
</tr>
</tbody>
</table>

**VB Type Listing**

```vb
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**

```delphi
type
    PEFaceNameType = array [0..PE_FACE_NAME_LEN-1] of Char;
    PEFontColorInfo = record
```

455
Structs

**C/VB/Delphi**

```c
struct PEFormulaSyntax {
    short StructSize;
    short formulaSyntax [PE_FS_SIZE];
}
```

**PEFormulaSyntax**

PEFormulaSyntax is used by PEGetFormulaSyntax, Page 292, and PESetFormulaSyntax, Page 394, to retrieve and set the syntax for the current and following formula API calls.

**C Syntax**

```c
#define PE_FS_SIZE 2
typedef struct PEFormulaSyntax {
    WORD StructSize;
    short formulaSyntax [PE_FS_SIZE];
} PEFormulaSyntax;
```

**Members**

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StructSize</td>
<td>Specifies the size of the PEFormulaSyntax structure. Initialize this member to PE_SIZEOF_FORMULA_SYNTAX.</td>
</tr>
<tr>
<td>formulaSyntax</td>
<td>Specifies the formula syntax. Use one of the PE_FST_XXX Formula Syntax Constants (page 543), or PE_UNCHANGED for no change.</td>
</tr>
</tbody>
</table>

**VB Syntax**

```vb
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

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StructSize</td>
<td>Specifies the size of the PEGeneralPrintWindowEventInfo structure. Initialize this member to PE_SIZEOF_GENERAL_PRINT_WINDOW_EVENT_INFO.</td>
</tr>
<tr>
<td>ignored</td>
<td>For 4 byte alignment. Ignore.</td>
</tr>
<tr>
<td>windowHandle</td>
<td>Frame window handle where the event happens.</td>
</tr>
</tbody>
</table>

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

```pascal
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

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;
  short dataAxisY2AutomaticDivision;
  short seriesAxisAutomaticDivision;
  long dataAxisYManualDivision;
  long dataAxisY2ManualDivision;
  long seriesAxisManualDivision;
  short dataAxisYAutoScale;
  short dataAxisY2AutoScale;
  short seriesAxisAutoScale;
} PEGraphAxisInfo;
```
Members

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StructSize</td>
<td>Specifies the size of the PEGraphAxisInfo structure. Initialize this member to PE_SIZEOF_GRAPH_AXIS_INFO.</td>
</tr>
<tr>
<td>groupAxisGridLine</td>
<td>Specifies GridLine option. Use one of the PE_GGT_XXX Chart Gridline Constants (page 534), or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td>dataAxisYGridLine</td>
<td>Specifies GridLine option. Use one of the PE_GGT_XXX Chart Gridline Constants (page 534), or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td>dataAxisY2GridLine</td>
<td>Specifies GridLine option. Use one of the PE_GGT_XXX Chart Gridline Constants (page 534), or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td>seriesAxisGridline</td>
<td>Specifies GridLine option. Use one of the PE_GGT_XXX Chart Gridline Constants (page 534), or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td>dataAxisYMinValue</td>
<td>Specifies the minimum value for the axis.</td>
</tr>
<tr>
<td>dataAxisYMaxValue</td>
<td>Specifies the maximum value for the axis.</td>
</tr>
<tr>
<td>dataAxisY2MinValue</td>
<td>Specifies the minimum value for the axis.</td>
</tr>
<tr>
<td>dataAxisY2MaxValue</td>
<td>Specifies the maximum value for the axis.</td>
</tr>
<tr>
<td>seriesAxisMinValue</td>
<td>Specifies the minimum value for the axis.</td>
</tr>
<tr>
<td>seriesAxisMaxValue</td>
<td>Specifies the maximum value for the axis.</td>
</tr>
<tr>
<td>dataAxisYNumberFormat</td>
<td>Specifies the format for the display of numeric values on the chart. Use one of the PE_GNF_XXX Chart Number Format Constants (page 535), or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td>dataAxisY2NumberFormat</td>
<td>Specifies the format for the display of numeric values on the chart. Use one of the PE_GNF_XXX Chart Number Format Constants (page 535), or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td>seriesAxisNumberFormat</td>
<td>Specifies the format for the display of numeric values on the chart. Use one of the PE_GNF_XXX Chart Number Format Constants (page 535), or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td>dataAxisYAutoRange</td>
<td>Boolean, or PE_UNCHANGED for no change. If TRUE, the axis will autorange.</td>
</tr>
<tr>
<td>dataAxisY2AutoRange</td>
<td>Boolean, or PE_UNCHANGED for no change. If TRUE, the axis will autorange.</td>
</tr>
<tr>
<td>seriesAxisAutoRange</td>
<td>Boolean, or PE_UNCHANGED for no change. If TRUE, the axis will autorange.</td>
</tr>
<tr>
<td>dataAxisYAutomaticDivision</td>
<td>PE_ADM_AUTOMATIC, PE_ADM_MANUAL, or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td>dataAxisY2AutomaticDivision</td>
<td>PE_ADM_AUTOMATIC, PE_ADM_MANUAL, or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td>seriesAxisAutomaticDivision</td>
<td>PE_ADM_AUTOMATIC, PE_ADM_MANUAL, or PE_UNCHANGED for no change.</td>
</tr>
</tbody>
</table>
VB Type Listing

Type PEGraphAxisInfo
    StructSize As Integer
    groupAxisGridLine As Integer
dataAxis1GridLine As Integer
    dataAxis1YGridLine As Integer
    seriesAxisGridLine As Integer
    dataAxis1Y2GridLine As Integer
    dataAxis1YMaxValue As Double
dataAxis1YMinValue As Double
    dataAxis1Y2MaxValue As Double
dataAxis1Y2MinValue As Double
    seriesAxisMinValue As Double
    seriesAxisMaxValue As Double
    dataAxis1YNumberFormat As Integer
    dataAxis1Y2NumberFormat As Integer
    seriesAxisNumberFormat As Integer
    dataAxis1YAutoRange As Integer
    dataAxis1Y2AutoRange As Integer
    seriesAxisAutoRange As Integer
    dataAxis1YAutomaticDivision As Integer
    dataAxis1Y2AutomaticDivision As Integer
    seriesAxisAutomaticDivision As Integer
dataAxis1YManualDivision As Long
dataAxis1Y2ManualDivision As Long
    seriesAxisManualDivision As Long
dataAxis1YAutoScale As Integer
    dataAxis1Y2AutoScale As Integer
    seriesAxisAutoScale As Integer
End Type

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataAxisYManualDivision</td>
<td>If the corresponding axis m_dataAxisYAutomaticDivision is</td>
</tr>
<tr>
<td></td>
<td>PE_ADM_AUTOMATIC, this field is ignored.</td>
</tr>
<tr>
<td>dataAxisY2ManualDivision</td>
<td>If the corresponding axis m_dataAxisY2AutomaticDivision is</td>
</tr>
<tr>
<td></td>
<td>PE_ADM_AUTOMATIC, this field is ignored.</td>
</tr>
<tr>
<td>seriesAxisManualDivision</td>
<td>If the corresponding axis m_seriesAxisAutomaticDivision is</td>
</tr>
<tr>
<td></td>
<td>PE_ADM_AUTOMATIC, this field is ignored.</td>
</tr>
<tr>
<td>dataAxisYAutoScale</td>
<td>Boolean, or PE_UNCHANGED for no change. If TRUE, the axis will autoscale.</td>
</tr>
<tr>
<td>dataAxisY2AutoScale</td>
<td>Boolean, or PE_UNCHANGED for no change. If TRUE, the axis will autoscale.</td>
</tr>
<tr>
<td>seriesAxisAutoScale</td>
<td>Boolean, or PE_UNCHANGED for no change. If TRUE, the axis will autoscale.</td>
</tr>
</tbody>
</table>
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, Page 295, and PESetGraphOptionInfo, Page 397.

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

<table>
<thead>
<tr>
<th>Structure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StructSize</td>
<td>Specifies the size of the PEGraphOptionsInfo structure. Initialize this member to PE_SIZEOF_GRAPH_OPTION_INFO.</td>
</tr>
<tr>
<td>graphColour</td>
<td>Use one of the PE_GCR_XXX Chart Options Constants (page 533), or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td>showLegend</td>
<td>BOOLEAN. Use TRUE to show the chart legend, FALSE to hide the chart legend, or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td>legendPosition</td>
<td>Use one of the PE_GLP_XXX Chart Options Constants (page 533), or PE_UNCHANGED for no change. Valid only if showLegend is TRUE.</td>
</tr>
<tr>
<td>pieSize</td>
<td>For pie charts and doughnut charts, use one of the PE_GPS_XXX Chart Pie Size Constants (page 536), or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td>detachedPieSlice</td>
<td>For pie charts and doughnut charts, use one of the PE_GDPS_XXX Chart Slice Detachment Constants (page 536), or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td>barSize</td>
<td>For bar charts, use one of the PE_GBS_XXX Chart Bar Size Constants (page 534), or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td>verticalBars</td>
<td>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.</td>
</tr>
<tr>
<td>markerSize</td>
<td>For line charts and bar charts, use one of the PE_GMS_XXX Chart Marker Size Constants (page 535), or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td>markerShape</td>
<td>For line charts and bar charts, use one of the PE_GMSP_XXX Chart Marker Shape Constants (page 535), or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td>dataPoints</td>
<td>Use one of the PE_GDP_XXX Chart Data Point Constants (page 534), or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td>dataValueNumberFormat</td>
<td>Use one of the PE_GNF_XXX Chart Number Format Constants (page 535), or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td>viewingAngle</td>
<td>For 3D charts, use one of the PE_GVA_XXX Chart Viewing Angle Constants (page 536), or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td>legendLayout</td>
<td>Specifies the legend layout. Use one of the PE_GLL_XXX Chart Legend Layout Constants (page 534).</td>
</tr>
</tbody>
</table>
**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**

```delphi
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

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StructSize</td>
<td>Specifies the size of the PEGraphTypeInfo structure. Initialize this member to PE_SIZEOF_GRAPH_TYPE_INFO.</td>
</tr>
<tr>
<td>graphType</td>
<td>Uses one of the PE_GT_XXX Graph Type Constants (page 547), or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td>graphSubtype</td>
<td>Uses one of the PE_GST_XXX Graph Subtype Constants (page 543), or PE_UNCHANGED for no change.</td>
</tr>
</tbody>
</table>

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, Page 301 to retrieve current options and by PESetGroupOptions, Page 402 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

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StructSize</td>
<td>Specifies the size of the PEGroupOptions structure. Initialize to PE_SIZEOF_GROUP_OPTIONS.</td>
</tr>
<tr>
<td>condition</td>
<td>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 (page 547), or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td>fieldName</td>
<td>Specifies the field name of the group field (of length PE_FIELD_NAME_LEN = 512). Use formula form or leave empty for no change.</td>
</tr>
<tr>
<td>sortDirection</td>
<td>Specifies one of the Sort Order Constants (page 551) or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td>repeatGroupHeader</td>
<td>TRUE, FALSE, or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td>keepGroupTogether</td>
<td>TRUE, FALSE, or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td>topOrBottomNGroups</td>
<td>Use one of the following PE_GO_TBN_XXX constants or PE_UNCHANGED for no change.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tbody>
</table>
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.

**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**

```vb
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

```delphi
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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StructSize</td>
<td>Specifies the size of the PEGroupTreeButtonClickedEventInfo structure. Initialize this member to PE_SIZEOF_GROUP_TREE_BUTTON_CLICKED_EVENT_INFO.</td>
</tr>
<tr>
<td>visible</td>
<td>Indicates whether the group tree is shown or hidden.</td>
</tr>
<tr>
<td>windowHandle</td>
<td>Frame window handle where the event happens.</td>
</tr>
</tbody>
</table>

VB Type Listing

```vbnet```
  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

```c
typedef struct PEHyperlinkEventInfo {
    WORD StructSize;
    WORD ignored;
    long windowHandle;
    char FAR *hyperlinkText;
} PEHyperlinkEventInfo;
```

Members

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StructSize</td>
<td>Specifies the size of the PEHyperlinkEventInfo structure. Initialize this member to PE_SIZEOF_HYPERLINKEVENTINFO.</td>
</tr>
<tr>
<td>ignored</td>
<td>For 4-byte alignment. Ignored.</td>
</tr>
<tr>
<td>windowHandle</td>
<td>Specifies HWND for the window in which the event occurred.</td>
</tr>
<tr>
<td>hyperlinkText</td>
<td>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.</td>
</tr>
</tbody>
</table>

PEJobInfo

Contains print job process and display information that is used by PEGetJobStatus, Page 304.

C Syntax

```c
typedef struct PEJobInfo {
    WORD StructSize;
    DWORD NumRecordsRead;
    DWORD NumRecordsSelected;
    DWORD NumRecordsPrinted;
    WORD DisplayPageN;
    WORD LatestPageN;
    WORD StartPageN;
    BOOL printEnded;
} PEJobInfo;
```
Members

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StructSize</td>
<td>Specifies the size of the PEJobInfo structure. Initialize to PE_SIZEOF_JOB_INFO.</td>
</tr>
<tr>
<td>NumRecordsRead</td>
<td>Specifies the number of records actually processed.</td>
</tr>
<tr>
<td>NumRecordsSelected</td>
<td>Specifies the number of records selected for inclusion in the report out of the total number of records read.</td>
</tr>
<tr>
<td>NumRecordsPrinted</td>
<td>Specifies the number of records actually printed or previewed.</td>
</tr>
<tr>
<td>DisplayPageN</td>
<td>Specifies the page number of the currently displayed page in the preview window.</td>
</tr>
<tr>
<td>LatestPageN</td>
<td>Specifies the page being generated. Once the printing is complete, this value is the number of the last page.</td>
</tr>
<tr>
<td>StartPageN</td>
<td>Specifies the number of the starting page. The value will normally be 1, but you can specify something else using PESetPrintOptions, Page 422.</td>
</tr>
<tr>
<td>printEnded</td>
<td>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.</td>
</tr>
</tbody>
</table>

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.

#### C Syntax

```c
typedef struct PELaunchSeagateAnalysisEventInfo {
    WORD StructSize;
    WORD ignored;
    long windowHandle;
    char FAR *pathFile;
} PELaunchSeagateAnalysisEventInfo;
```

#### Members

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StructSize</td>
<td>Specifies the size of the PELaunchSeagateAnalysisEventInfo structure. Initialize this member to PE_SIZEOF_LAUNCH_SEAGATE_ANALYSIS_EVENT_INFO.</td>
</tr>
<tr>
<td>ignored</td>
<td>For 4-byte alignment. Ignored.</td>
</tr>
<tr>
<td>windowHandle</td>
<td>Specifies HWND for the window in which the event occurred.</td>
</tr>
<tr>
<td>pathFile</td>
<td>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.</td>
</tr>
</tbody>
</table>

### PELogOnInfo

PELogOnInfo contains log on information that is used by PEGetNthTableLogOnInfo, Page 332; PESetNthTableLogOnInfo, Page 414; PEOpenOnServer, Page 359; and PEOpenOffServer, Page 358 for logging on and off SQL and password-protected non-SQL databases.

#### C Syntax

```c
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

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StructSize</td>
<td>Specifies the size of the PELogOnInfo structure. Initialize this member to PE_SIZEOF_LOGON_INFO.</td>
</tr>
<tr>
<td>ServerName</td>
<td>Specifies the logon name for the server (of length PE_SERVERNAME_LEN = 128, NULL-terminated) used to create the report. See Remarks below.</td>
</tr>
</tbody>
</table>
Remarks

- All strings must be null-terminated.
- Password is undefined when getting information from the report.
- Password must be set using `PEGetNthTableLogOnInfo`. Page 332. 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;
  end;
```
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. |

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_MOUSE_NOTSUPPORTED</td>
<td></td>
</tr>
<tr>
<td>PE_MOUSE_DOWN</td>
<td></td>
</tr>
<tr>
<td>PE_MOUSE_UP</td>
<td></td>
</tr>
<tr>
<td>PE_MOUSE_DOUBLE_CLICK</td>
<td>For Mouse Left click or Mouse Middle click.</td>
</tr>
</tbody>
</table>

| clickFlags | Indicates the source of the event, which can be any combination of the following PE_CF_XXX virtual key state-mask constants. |

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_CF_NONE</td>
<td>0x0000</td>
<td>No button pressed</td>
</tr>
<tr>
<td>PE_CF_LBUTTON</td>
<td>0x0001</td>
<td>Left mouse button</td>
</tr>
<tr>
<td>PE_CF_RBUTTON</td>
<td>0x0002</td>
<td>Right mouse button</td>
</tr>
</tbody>
</table>
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;
PEOleObjectInfo

PEOleObjectInfo contains type, update method, and path (linked objects only) information for OLE objects.

C Syntax

typedef struct PEOleObjectInfo {
    WORD StructSize;
    WORD oleObjectType;
    char linkSource [PE_OOI_PATHLEN];
    WORD updateType;
} PEOleObjectInfo;

Members

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StructSize</td>
<td>Specifies the size of the PEOleObjectInfo structure. Initialize this member to PE_SIZEOF_OLE_OBJECT_INFO.</td>
</tr>
<tr>
<td>oleObjectType</td>
<td>Specifies the type of OLE object. Use one of the PE_OOI_XXX Ole Object Type Constants (page 550).</td>
</tr>
<tr>
<td>linkSource</td>
<td>Specifies the source of the OLE link (of length PE_OOI_PATHLEN = 256). This member is valid only for linked objects.</td>
</tr>
<tr>
<td>updateType</td>
<td>Specifies the update method of the OLE object. Use one of the PE_OOI_XXX Ole Object Update Constants (page 550).</td>
</tr>
</tbody>
</table>

PEParameterFieldInfo

The PEParameterFieldInfo structure contains information related to parameter fields in a report. This structure is used by PEGetNthParameterField, Page 324 to get information about a specific parameter field and by PESetNthParameterField, Page 409 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. CRWNULL is of Type String and is independent of the data type of the parameter.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StructSize</td>
<td>Specifies the size of the PEParameterFieldInfo structure. Initialize this member to PE_SIZEOF_PARAMETER_FIELD_INFO.</td>
</tr>
<tr>
<td>ValueType</td>
<td>Specifies the data type of the parameter field. The Seagate Crystal Report Engine supports the data types and associated PE_PF_XXX Parameter Field Value Type Constants (page 550).</td>
</tr>
<tr>
<td>DefaultValueSet</td>
<td>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.</td>
</tr>
<tr>
<td>CurrentValueSet</td>
<td>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.</td>
</tr>
<tr>
<td>Name</td>
<td>Specifies the name of the parameter field (of length PE_PF_NAME_LEN = 256, NULL-terminated) to be assigned a new value.</td>
</tr>
<tr>
<td>Prompt</td>
<td>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.</td>
</tr>
</tbody>
</table>
### 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.
  
  - For new application development, the Default/CurrentValueSet must be set to FALSE, so that the Default/CurrentValue members of PEPParameterFieldInfo 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, Page 262
  PEAddParameterCurrentValue, Page 263
  ```

---

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DefaultValue</td>
<td>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).</td>
</tr>
<tr>
<td>CurrentValue</td>
<td>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).</td>
</tr>
<tr>
<td>ReportName</td>
<td></td>
</tr>
</tbody>
</table>
For existing applications only; not for new development. See Remarks below. The name of the report (of length PE_PF_REPORT_NAME_LEN = 128) where the field belongs (used only with PEGetNthParameterField, Page 324). |
| 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, Page 336, and PESetParameterMinMaxValue, Page 417. 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, Page 336, and PESetParameterMinMaxValue, Page 417. 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. |
To determine if a parameter field is a stored procedure, use PEGetNthParameterType, Page 325, or PEGetNthParameterField, Page 324.

**VB type listing**

```vb
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**

```delphi
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;

    PEPParameterFieldInfo = record
        structSize: Word;
        ValueType: Word;
```
typedef struct PEParameterPickListOption {
    WORD StructSize;
    short showDescOnly;
    short sortMethod;
    short sortBasedOnDesc;
} PEParameterPickListOption;

Members

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StructSize</td>
<td>Specifies the size of the PEParameterFieldInfo structure. Initialize this member to PE_SIZEOF_PICK_LIST_OPTION.</td>
</tr>
<tr>
<td>showDescOnly</td>
<td>Boolean, or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td>sortMethod</td>
<td>Use one of the PE_OR_XXX Sort Method Constants (page 551), or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td>sortBasedOnDesc</td>
<td>Boolean, or PE_UNCHANGED for no change.</td>
</tr>
</tbody>
</table>

VB Type Listing

Type PEParameterPickListOption
    StructSize As Integer
    showDescOnly As Integer
    sortMethod As Integer
    sortBasedOnDesc As Integer
End Type
Delphi Record Listing

type
    PEPParameterPickListOption = 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*, Page 22.

C Syntax

typedef struct PEPParameterValueInfo {
    WORD StructSize;
    short isNullable;
    short disallowEditing;
    short allowMultipleValues;
    short hasDiscreteValues;
    short partOfGroup;
    short groupNum;
    short mutuallyExclusiveGroup
} PEPParameterValueInfo;

Members

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StructSize</td>
<td>Specifies the size of the PEPParameterValueInfo structure. Set this member to PE_SIZEOF_PARAMETER_VALUE_INFO</td>
</tr>
<tr>
<td>isNullable</td>
<td>Specifies whether or not the parameter field can be set to NULL. Set to TRUE, FALSE or PE_UNCHANGED if no change.</td>
</tr>
<tr>
<td>disallowEditing</td>
<td>Indicates whether the parameter field value can be edited. Set to TRUE, FALSE, or PE_UNCHANGED if no change.</td>
</tr>
<tr>
<td>allowMultipleValues</td>
<td>Specifies whether or not the parameter field can contain multiple values. Set to TRUE, FALSE or PE_UNCHANGED if no change.</td>
</tr>
<tr>
<td>hasDiscreteValues</td>
<td>Specifies whether or not the parameter field contains discreet values, range values, or both. Uses one of the following PE_DR_XXX Constants. See <em>Working with Parameter Values and Ranges</em>, Page 21.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_DR_HASRANGE</td>
<td>Only ranges are present.</td>
</tr>
<tr>
<td>PE_DR_HASDISCRETE</td>
<td>Only discrete values are present.</td>
</tr>
</tbody>
</table>
VB Type Listing

Type PParameterValueInfo
    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, Page 340, to retrieve current options and PESetPrintOptions, Page 422, 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;
};
Members

<table>
<thead>
<tr>
<th>StructSize</th>
<th>Specifies the size of the PEPrintOptions structure. Initialize this member to PE_SIZEOF_PRINT_OPTIONS.</th>
</tr>
</thead>
<tbody>
<tr>
<td>startPageN</td>
<td>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>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>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>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>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<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>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>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>Specifies a path and file name (of length PE_FILE_PATH_LEN = 512) if the report is printed to a file.</td>
</tr>
</tbody>
</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

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StructSize</td>
<td>Specifies the size of the PEReadingRecordsEventInfo structure. Initialize this member to PE_SIZEOF_READING_RECORDS_EVENT_INFO.</td>
</tr>
<tr>
<td>cancelled</td>
<td>Boolean value indicates whether the reading records is canceled.</td>
</tr>
<tr>
<td>recordsRead</td>
<td>Indicates how many records have been read.</td>
</tr>
<tr>
<td>recordsSelected</td>
<td>Indicates how many records have been selected.</td>
</tr>
<tr>
<td>done</td>
<td>Boolean value indicates whether reading records is finished.</td>
</tr>
</tbody>
</table>

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

```delphi
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

```c
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>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StructSize</td>
<td>Specifies the size of the PEReportFieldMappingInfo structure. Initialize this member to PE_SIZEOF_REPORT_FIELDMAPPING_INFO.</td>
</tr>
<tr>
<td>valueType</td>
<td>Indicates the field value type. The Value Type can be one of the following constants:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_FVT_INT8SFIELD</td>
<td>8-bit integer signed</td>
</tr>
<tr>
<td>PE_FVT_INT8UFIELD</td>
<td>8-bit integer unsigned</td>
</tr>
<tr>
<td>PE_FVT_INT16SFIELD</td>
<td>16-bit integer signed</td>
</tr>
<tr>
<td>PE_FVT_INT16UFIELD</td>
<td>16-bit integer unsigned</td>
</tr>
<tr>
<td>PE_FVT_INT32SFIELD</td>
<td>32-bit integer signed</td>
</tr>
<tr>
<td>PE_FVT_INT32UFIELD</td>
<td>32-bit integer unsigned</td>
</tr>
<tr>
<td>PE_FVT_NUMBERFIELD</td>
<td>Number field</td>
</tr>
</tbody>
</table>
**Structure of Delphi Record Listing**

<table>
<thead>
<tr>
<th>Delphi Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_FVT_CURRENCYFIELD</td>
<td>Currency field</td>
</tr>
<tr>
<td>PE_FVT_BOOLEANFIELD</td>
<td>Boolean field</td>
</tr>
<tr>
<td>PE_FVT_DATEFIELD</td>
<td>Date field</td>
</tr>
<tr>
<td>PE_FVT_TIMEFIELD</td>
<td>Time field</td>
</tr>
<tr>
<td>PE_FVT_STRINGFIELD</td>
<td>String field</td>
</tr>
<tr>
<td>PE_FVT_TRANSIENTMEMOFIELD</td>
<td>Transient Memo field</td>
</tr>
<tr>
<td>PE_FVT_PERSISTENTMEMOFIELD</td>
<td>Persistent Memo field</td>
</tr>
<tr>
<td>PE_FVT_BLOBFIELD</td>
<td>BLOB field</td>
</tr>
<tr>
<td>PE_FVT_DATETIMEFIELD</td>
<td>Date/Time field</td>
</tr>
<tr>
<td>PE_FVT_BITMAPFIELD</td>
<td>Bitwise field</td>
</tr>
<tr>
<td>PE_FVT_ICONFIELD</td>
<td>Icon field</td>
</tr>
<tr>
<td>PE_FVT_PICTUREFIELD</td>
<td>Picture field</td>
</tr>
<tr>
<td>PE_FVT_OLEFIELD</td>
<td>OLE field</td>
</tr>
<tr>
<td>PE_FVT_GRAPHFIELD</td>
<td>Graph field</td>
</tr>
<tr>
<td>PE_FVT_BLOBFIELD</td>
<td>BLOB field</td>
</tr>
</tbody>
</table>

**Field Descriptions**

- **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` (page 452), which contains the list of database fields. If the field is unmapped then this value is -1.

**Delphi Record Listing**

```delphi
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;
```
PEReportOptions

PEReportOptions contains report option information. This information is used by PEGetReportOptions, Page 341, to retrieve current options and by PESetReportOptions, Page 423, 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 wysiwygMode;
} PEReportOptions;

Members

For each member, except as noted below, use TRUE, FALSE, or PE_UNCHANGED for no change.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>structSize</td>
<td>Specifies the size of the PEReportOptions structure. Initialize to PE_SIZEOF_REPORT_OPTIONS.</td>
</tr>
<tr>
<td>saveDataWithReport</td>
<td>Specifies whether or not data should be saved with the report. Boolean, or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td>saveSummariesWithReport</td>
<td>Specifies whether to save summaries with the report. Boolean, or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td>useIndexForSpeed</td>
<td>Specifies whether or not to use index values. Boolean, or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td>translateDOSStrings</td>
<td>Specifies whether or not to translate DOS strings. Boolean, or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td><strong>Constant</strong></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
</tr>
<tr>
<td><strong>translateDOSMemos</strong></td>
<td>Specifies whether or not to translate DOS memos. Boolean, or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td><strong>convertDateTimeType</strong></td>
<td>Specifies whether or not to convert DATE/Time format to another format. Use one of the following constants, or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td><strong>Constant</strong></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td>PE_RPTOPT_CVTDATETIMETO炕</td>
<td>0</td>
</tr>
<tr>
<td>PE_RPTOPT_CVTDATETIMETODATE</td>
<td>1</td>
</tr>
<tr>
<td>PE_RPTOPT_KEEPDATETIMETYPE</td>
<td>2</td>
</tr>
<tr>
<td><strong>convertNullFieldToDefault</strong></td>
<td>Specifies whether or not to convert NULL parameter fields to their default values. Boolean, or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td><strong>morePrintEngineError Messages</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>caseInsensitiveSQLData</strong></td>
<td>Specifies whether or not to perform a case insensitive search for SQL data. Boolean, or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td><strong>verifyOnEveryPrint</strong></td>
<td>Specifies whether or not to perform database verification for every print job. Boolean, or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td><strong>zoomMode</strong></td>
<td>Use one of the <strong>Zoom Level Constants</strong> (page 553), or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td><strong>hasGroupTree</strong></td>
<td>Specifies whether or not there is a group tree associated with the report. Boolean, or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td><strong>dontGenerateData ForHiddenObjects</strong></td>
<td>Specifies whether or not to generate data for hidden objects. Boolean, or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td><strong>performGroupingOnServer</strong></td>
<td>Specifies whether or not to perform grouping on servers. Boolean, or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td><strong>doAsyncQuery</strong></td>
<td>Boolean, or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td><strong>promptMode</strong></td>
<td>Specifies the prompt mode. Use one of the following constants, or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td><strong>Constant</strong></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td>PE_RPTOPT_PROMPT_NONE</td>
<td>0</td>
</tr>
<tr>
<td>PE_RPTOPT_PROMPT_NORMAL</td>
<td>1</td>
</tr>
<tr>
<td>PE_RPTOPT_PROMPT_ALWAYS</td>
<td>2</td>
</tr>
<tr>
<td><strong>selectDistinctRecords</strong></td>
<td>Specifies whether or not to select distinct records. Boolean, or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td><strong>wysiwygMode</strong></td>
<td>Specifies whether or not to use wysiwyg mode. Boolean, or PE_UNCHANGED for no change.</td>
</tr>
</tbody>
</table>
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
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;
    end;
PEReportSummaryInfo

PEReportSummaryInfo contains report summary information, corresponding to the report summary information in the Seagate Crystal Reports Designer.

C Syntax

```c
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

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StructSize</td>
<td>Specifies the size of the structure. Initialize this member to PE_SIZEOF_REPORT_SUMMARY_INFO.</td>
</tr>
<tr>
<td>applicationName</td>
<td>Specifies an application name (of length PE_APPLICATION_NAME_LEN = 128) for the application using this report. This member is read only.</td>
</tr>
<tr>
<td>title</td>
<td>Specifies the title (of length PE_TITLE_LEN = 128) of the current report.</td>
</tr>
<tr>
<td>subject</td>
<td>Specifies the subject (of length PE_SI_SUBJECT_LEN = 128) of the current report.</td>
</tr>
<tr>
<td>author</td>
<td>Specifies the author (of length PE_SI_AUTHOR_LEN = 128) of the current report.</td>
</tr>
<tr>
<td>keywords</td>
<td>Specifies the keywords (of length PE_SI_KEYWORDS_LEN = 128) included for the current report.</td>
</tr>
<tr>
<td>comments</td>
<td>Specifies any comments (of length PE_SI_COMMENTS_LEN = 512) for the current report.</td>
</tr>
<tr>
<td>reportTemplate</td>
<td>Specifies the report template (of length PE_SI_REPORT_TEMPLATE_LEN = 128) for the current report.</td>
</tr>
<tr>
<td>savePreviewPicture</td>
<td>Specifies whether or not to save the preview picture. Boolean or PE_UNCHANGED for no change.</td>
</tr>
</tbody>
</table>
**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

```c
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

```vb
Type PESearchButtonClickedEventInfo
    windowHandle As Long
    searchString As String * PE_SEARCH_STRING_LEN
    StructSize As Integer
End Type
```

Delphi Record Listing

```delphi
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, Page 344 and PEGetAreaFormat, Page 285 to retrieve current settings and PESetSectionFormat, Page 426 and PESetAreaFormat, Page 377 to pass new settings.

C Syntax

typedef struct PESectionOptions {
    WORD StructSize;
    WORD short visible;
    WORD newPageBefore;
    WORD newPageAfter;
    WORD keepTogether;
    WORD suppressBlankSection;
    WORD resetPageNAfter;
    WORD printAtBottomOfPage;
    COLORREF backgroundColor;
    short underlaySection;
    short showArea;
    short freeFormPlacement;
    short reserveMinimumPageFooter;
} PESectionOptions;

Members

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StructSize</td>
<td>Specifies the size of the PESectionOptions structure. Initialize to PE_SIZEOF_SECTION_OPTIONS.</td>
</tr>
<tr>
<td>visible</td>
<td>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.</td>
</tr>
<tr>
<td>newPageBefore</td>
<td>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.</td>
</tr>
<tr>
<td>newPageAfter</td>
<td>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.</td>
</tr>
<tr>
<td>keepTogether</td>
<td>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.</td>
</tr>
<tr>
<td>suppressBlankSection</td>
<td>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.</td>
</tr>
</tbody>
</table>
**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`.

- `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.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>resetPageNAfter</strong></td>
<td>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 <code>TRUE</code> to reset the page number, <code>FALSE</code> to retain the standard numbering, or <code>PE_UNCHANGED</code> for no change.</td>
</tr>
<tr>
<td><strong>printAtBottomOfPage</strong></td>
<td>Specifies whether or not the program is to cause each group value to print only at the bottom of a page; <code>FALSE</code> to have the values print in their normal position, or <code>PE_UNCHANGED</code> for no change.</td>
</tr>
<tr>
<td><strong>backgroundColor</strong></td>
<td>Specifies the RGB color value contained in the <code>COLORREF</code> (page 523) value, for section formats only. Use <code>PE_UNCHANGED_COLOR</code> to preserve the existing color or <code>PE_NO_COLOR</code> for no color.</td>
</tr>
<tr>
<td><strong>underlaySection</strong></td>
<td>Indicates whether or not the specified section is to underlay the following section(s). <code>TRUE</code>, <code>FALSE</code>, or <code>PE_UNCHANGED</code> for no change. The user can drill down on a hidden area.</td>
</tr>
<tr>
<td><strong>showArea</strong></td>
<td>Specifies <code>TRUE</code> to show an area, <code>FALSE</code> to hide an area, or <code>PE_UNCHANGED</code> for no change. The user can drill down on a hidden area.</td>
</tr>
<tr>
<td><strong>freeFormPlacement</strong></td>
<td>Design time flag. If set to <code>TRUE</code>, an object can be placed anywhere in a section. Use <code>PE_UNCHANGED</code> for no change.</td>
</tr>
<tr>
<td><strong>reserveMinimumPageFooter</strong></td>
<td>Used to reduce unnecessary white space in the page footer area containing more than one conditionally formatted section. When set to <code>TRUE</code>, the space required to display only one section (the tallest) is reserved. When set to <code>FALSE</code> (default), the maximum height necessary to display every section in the page footer area at full height will be reserved. See Remarks below.</td>
</tr>
</tbody>
</table>
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, Page 333 to retrieve current information and PESetNthTableSessionInfo, Page 416 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

<table>
<thead>
<tr>
<th>Members</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StructSize</td>
<td>Specifies the size of the PESessionInfo structure. Initialize this member to PE_SIZEOF_SESSION_INFO.</td>
</tr>
<tr>
<td>UserID</td>
<td>Specifies the user ID value (of length PE_SESS_USERID_LEN = 128) needed for logging on to the system.</td>
</tr>
<tr>
<td>Password</td>
<td>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.</td>
</tr>
<tr>
<td>SessionHandle</td>
<td>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.</td>
</tr>
</tbody>
</table>

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, Page 416.
- 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 (page 470) and calling PESetNthTableLogOnInfo, Page 414.
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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StructSize</td>
<td>Specifies the size of the PEShowGroupEventInfo structure. Initialize this member to PE_SIZEOF_SHOW_GROUP_EVENT_INFO.</td>
</tr>
<tr>
<td>groupLevel</td>
<td>The number of the group name in the group list.</td>
</tr>
<tr>
<td>windowHandle</td>
<td>Frame window handle where the event happens.</td>
</tr>
<tr>
<td>groupList</td>
<td>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.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StructSize</td>
<td>Specifies the size of the PEStartEventInfo structure. Initialize this member to PE_SIZEOF_START_EVENT_INFO.</td>
</tr>
<tr>
<td>destination</td>
<td>Specifies the process destination. Uses one of the PE_TO_XXX Job Destination Constants (page 549).</td>
</tr>
</tbody>
</table>

VB Type Listing

Type PEStartEventInfo
    StructSize As Integer
    destination As Integer
End Type
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

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StructSize</td>
<td>Specifies the size of the PEStopEventInfo structure. Initialize this member to PE_SIZEOF_STOP_EVENT_INFO.</td>
</tr>
<tr>
<td>destination</td>
<td>Specifies the process destination. Uses one of the PE_TO_XXX Job Destination Constants (page 549).</td>
</tr>
<tr>
<td>jobStatus</td>
<td>Indicates the current status of the job. Uses one of the PE_JOBXXX Job Status Constants (page 549).</td>
</tr>
</tbody>
</table>

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 contains information about subreports in a report. This structure is used by PEGetSubreportInfo, Page 352, to gather information about a specified subreport.

**C Syntax**

```c
typedef struct PESubreportInfo {
    WORD StructSize;
    char name [PE_SUBREPORT_NAME_LEN];
    short NLinks;
    short isOnDemand;
    short external;
    short reimportOption;
} PESubreportInfo;
```

**Members**

<table>
<thead>
<tr>
<th>StructSize</th>
<th>Specifies the size of the PESubreportInfo structure. Initialize this member to PE_SIZEOF_SUBREPORT_INFO.</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>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.</td>
</tr>
<tr>
<td>NLinks</td>
<td>Specifies the number of links to primary report data.</td>
</tr>
<tr>
<td>isOnDemand</td>
<td>TRUE if the subreport is a real-time subreport. FALSE otherwise.</td>
</tr>
<tr>
<td>external</td>
<td>TRUE if the subreport is imported. FALSE otherwise.</td>
</tr>
<tr>
<td>reimportOption</td>
<td>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).</td>
</tr>
</tbody>
</table>

**VB Type Listing**

```vb
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;
      NLLinks: smallint
      IsOnDemand: smallint
  end;

PETableDifferenceInfo

Read-Only Struct PETableDifferenceInfo contains database table information that is used by
PECheckNthTableDifferences, Page 268.

C Syntax

typedef struct PETableDifferenceInfo {
     WORD StructSize;
     DWORD tableDifferences;
     DWORD reserved1;
     DWORD reserved2;
} PETableDifferenceInfo;

Members

<table>
<thead>
<tr>
<th>StructSize</th>
<th>Specifies the size of the PETableLocation structure. Initialize this member to PE_SIZEOF_TABLE_DIFFERENCE_INFO.</th>
</tr>
</thead>
<tbody>
<tr>
<td>tableDifferences</td>
<td>Read-Only. Returns any combination of the following PE_TCD_XXX TableDifference Constants.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_TCD_OKAY</td>
<td>0x00000000</td>
</tr>
<tr>
<td>PE_TCD_DATABASENOTFOUND</td>
<td>0x00000001</td>
</tr>
<tr>
<td>PE_TCD_SERVERNOTFOUND</td>
<td>0x00000002</td>
</tr>
<tr>
<td>PE_TCD_SERVERNOTOPENED</td>
<td>0x00000004</td>
</tr>
<tr>
<td>PE_TCD_ALIASCHANGED</td>
<td>0x00000008</td>
</tr>
<tr>
<td>PE_TCD_INDEXESCHANGED</td>
<td>0x00000010</td>
</tr>
<tr>
<td>PE_TCD_DRIVERCHANGED</td>
<td>0x00000020</td>
</tr>
<tr>
<td>PE_TCD_DICTIONARYCHANGED</td>
<td>0x00000040</td>
</tr>
</tbody>
</table>
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;

    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_FIELDDOTHER      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.
PETableLocation

PETableLocation contains database location information that is used with the PEGetNthTableLocation, Page 331 to gather current location information and PESetNthTableLocation, Page 413 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

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StructSize</td>
<td>Specifies the size of the PETableLocation structure. Initialize this member to PE_SIZEOF_TABLE_LOCATION.</td>
</tr>
<tr>
<td>Location</td>
<td>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.</td>
</tr>
</tbody>
</table>

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>

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SubLocation</td>
<td>Specifies the database sublocation (of length PE_TABLE_LOCATION_LEN = 256, NULL-terminated).</td>
</tr>
<tr>
<td>ConnectBuffer</td>
<td>Specifies the connection buffer (of length PE_CONNECTION_BUFFER_LEN = 512, NULL-terminated) containing connection information for attached tables.</td>
</tr>
</tbody>
</table>
**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**

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StructSize</td>
<td>Specifies the size of the PETablePrivateInfo structure. Initialize this member to PE_SIZEOF_TABLE_PRIVATE_INFO.</td>
</tr>
<tr>
<td>nBytes</td>
<td>Specifies the length of the data starting at the dataPtr.</td>
</tr>
<tr>
<td>tag</td>
<td>Specifies 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.</td>
</tr>
<tr>
<td>dataPtr</td>
<td>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.</td>
</tr>
</tbody>
</table>
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, Page 334.

C Syntax

typedef struct PETableType {
  WORD StructSize;
  char DLLName [PE_DLL_NAME_LEN];
  char DescriptiveName [PE_FULL_NAME_LEN];
  WORD DbType;
} PETableType;

Members

| StructSize | Specifies the size of the PETableType structure. Initialize this member to PE_SIZEOF_TABLE_TYPE. |
| DLLName | 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: |

<table>
<thead>
<tr>
<th>Use this DLL</th>
<th>For this standard non-SQL database</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDBBDE.DLL</td>
<td>Borland Database Engine</td>
</tr>
<tr>
<td>PDBBNBD.DLL</td>
<td>Bound reports</td>
</tr>
<tr>
<td>PDBDAO.DLL</td>
<td>DAO data sources (Access)</td>
</tr>
<tr>
<td>PDBJET.DLL</td>
<td>Access</td>
</tr>
<tr>
<td>PD8PDX.DLL</td>
<td>Paradox</td>
</tr>
<tr>
<td>PDBXBBSE.DLL</td>
<td>dBASE, FoxPro, Clipper</td>
</tr>
<tr>
<td>PDCTBTRV.DLL</td>
<td>Btrieve</td>
</tr>
</tbody>
</table>
Use this DLL | For this SQL database
---|---
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 (page 537).

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, Page 353, and PESetTrackCursorInfo, Page 432.
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 (page 552), or PE_UNCHANGED for no change.
- Each member of type long is currently reserved and should not be used.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StructSize</td>
<td>Specifies the size of the PETrackCursorInfo structure. Initialize this member to PE_SIZEOF_TRACK_CURSOR_INFO.</td>
</tr>
<tr>
<td>groupAreaCursor</td>
<td>Specifies the group area cursor. Use one of the PE_TC_XXX Track Cursor Constants (page 552) or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td>groupAreaFieldCursor</td>
<td>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 (page 552) or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td>detailAreaCursor</td>
<td>Specifies the Details area cursor. Use one of the PE_TC_XXX Track Cursor Constants (page 552) or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td>detailAreaFieldCursor</td>
<td>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 (page 552) or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td>graphCursor</td>
<td>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 (page 552) or PE_UNCHANGED for no change.</td>
</tr>
<tr>
<td>groupAreaCursorHandle</td>
<td>Reserved, do not use.</td>
</tr>
<tr>
<td>groupAreaFieldCursorHandle</td>
<td>Reserved, do not use.</td>
</tr>
<tr>
<td>detailAreaCursorHandle</td>
<td>Reserved, do not use.</td>
</tr>
</tbody>
</table>
Remarks

- By default, all the cursors are `PE_TC_ARROW_CURSOR`. If the `canDrillDown` option in `PEWindowOptions` (page 509), is True, then `groupAreaCursor`, `groupAreaFieldCursor`, and `graphCursor` will be `PE_TC_MAGNIFY_CURSOR`.

### VB Type Listing

```vb
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

```delphi
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, Page 275 to return converted parameter values in simple types and by PEConvertVInfoToPFInfo, Page 276 to accept values for conversion to the binary format required by PESetNthParameterField, Page 409.

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

<table>
<thead>
<tr>
<th>StructSize</th>
<th>Specifies the size of the PEValueInfo structure. Initialize this member to PE_SIZEOF_VALUE_INFO.</th>
</tr>
</thead>
<tbody>
<tr>
<td>valueType</td>
<td>Specifies the data type of the parameter field. The Seagate Crystal Report Engine supports the following data types and associated PE_VI_XXX Value Type constants.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>PE_VI_NUMBER</td>
</tr>
<tr>
<td>Currency</td>
<td>PE_VI_CURRENCY</td>
</tr>
<tr>
<td>Boolean</td>
<td>PE_VI_BOOLEAN</td>
</tr>
<tr>
<td>Date</td>
<td>PE_VI_DATE</td>
</tr>
<tr>
<td>String</td>
<td>PE_VI_STRING</td>
</tr>
<tr>
<td>DateTime</td>
<td>PE_VI_DATETIME</td>
</tr>
<tr>
<td>Time</td>
<td>PE_VI_TIME</td>
</tr>
</tbody>
</table>
### VB Type Listing

```vbnet
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
```

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>viNumber</td>
<td>Specifies the value if the parameter is a numeric value.</td>
</tr>
<tr>
<td>viCurrency</td>
<td>Specifies the value if the parameter is a currency value.</td>
</tr>
<tr>
<td>viBoolean</td>
<td>Specifies the value if the parameter is a Boolean value.</td>
</tr>
<tr>
<td>viString</td>
<td>Specifies the value if the parameter is a string value (of length PE_VI_STRING_LEN = 256).</td>
</tr>
<tr>
<td>viDate</td>
<td>Specifies the value if the parameter is a Date value (year, month, day).</td>
</tr>
<tr>
<td>viDateTime</td>
<td>Specifies the value if the parameter is a Date/Time value (year, month, day, hour, minute, second).</td>
</tr>
<tr>
<td>viTime</td>
<td>Specifies the value if the parameter is a Time value (hour, minute, second).</td>
</tr>
<tr>
<td>viColor</td>
<td>Specifies the value if the parameter is a color value.</td>
</tr>
<tr>
<td>viInteger</td>
<td>Specifies the value if the parameter is a integer value.</td>
</tr>
<tr>
<td>viC</td>
<td>Specifies the value if the parameter is a char value.</td>
</tr>
<tr>
<td>ignored</td>
<td>Internal use only for 4 byte alignment. Do not use.</td>
</tr>
<tr>
<td>viLong</td>
<td>Specifies the value if the parameter is a long value.</td>
</tr>
</tbody>
</table>
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;

PEWindowOptions contains information related to preview window options. This structure is used by
PEGetWindowOptions, Page 356, and PESetWindowOptions, Page 433, 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.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StructSize</td>
<td>Specifies the size of the P EW indowOptions structure. Initialize this member to PE_SIZEOF_WINDOW_OPTIONS.</td>
</tr>
<tr>
<td>hasGroupTree</td>
<td>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.</td>
</tr>
<tr>
<td>canDrillDown</td>
<td>Specifies whether or not drill-down capabilities will be active in the window. By default, this value is set to False.</td>
</tr>
<tr>
<td>hasNavigationControls</td>
<td>Specifies whether or not navigation controls will appear in the window. By default, this value is set to True.</td>
</tr>
<tr>
<td>hasCancelButton</td>
<td>Specifies whether or not a Cancel button will appear in the window. By default, this value is set to True.</td>
</tr>
<tr>
<td>hasPrintButton</td>
<td>Specifies whether or not a Print button will appear in the window. By default, this value is set to True.</td>
</tr>
<tr>
<td>hasExportButton</td>
<td>Specifies whether or not an Export button will appear in the window. By default, this value is set to True.</td>
</tr>
<tr>
<td>hasZoomControl</td>
<td>Specifies whether or not zoom controls will appear in the window. By default, this value is set to True.</td>
</tr>
<tr>
<td>hasCloseButton</td>
<td>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 turn hasCloseButton on unless you set it to False.</td>
</tr>
<tr>
<td>hasProgressControls</td>
<td>Specifies whether or not progress controls will appear in the window. By default, this value is set to True.</td>
</tr>
<tr>
<td>hasSearchButton</td>
<td>Specifies whether or not a Search button will appear in the window. By default, this value is set to False.</td>
</tr>
<tr>
<td>hasPrintSetupButton</td>
<td>Specifies whether or not a Print Setup button will appear in the window.</td>
</tr>
<tr>
<td>hasRefreshButton</td>
<td>Specifies whether or not a Refresh button will appear in the window.</td>
</tr>
<tr>
<td>showToolbarTips</td>
<td>Specifies whether or not Tooltips are shown on the Toolbar. By default, this value is set to True (default is visible tooltips on toolbar).</td>
</tr>
<tr>
<td>showDocumentTips</td>
<td>Specifies whether or not Tooltips are shown on the Document. By default, this value is set to False (default is hidden tooltips on document).</td>
</tr>
<tr>
<td>hasLaunchButton</td>
<td>Specifies whether or not a button to launch Seagate Analysis is placed on the toolbar. By default, this value is set to False.</td>
</tr>
</tbody>
</table>
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 Record 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**

```c
typedef struct PEZoomLevelChangingEventInfo {
    WORD StructSize;
    WORD zoomLevel;
    long windowHandle;
} PEZoomLevelChangingEventInfo;
```

**Members**

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StructSize</td>
<td>Specifies the size of the PEZoomLevelChangingEventInfo structure. Initialize this member to PE_SIZEOF_ZOOM_LEVEL_CHANGING_EVENT_INFO.</td>
</tr>
<tr>
<td>zoomLevel</td>
<td>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 (page 553).</td>
</tr>
<tr>
<td>windowHandle</td>
<td>Specifies the frame window handle where the event happens.</td>
</tr>
</tbody>
</table>

**VB Type Listing**

```vb
Type PEZoomLevelChangingEventInfo
    StructSize As Integer
    zoomLevel As Integer
    windowHandle As Long
End Type
```

**Delphi Record Listing**

```delphi
type
    PEZoomLevelChangingEventInfo = record
        StructSize: Word;
        zoomLevel: Word;
        windowHandle: HWND;
    end;
```

**UXDDiskOptions**

UXDDiskOptions contains file name information that is used by PEEnableEventInfo (page 446), when you want to export to a disk file.

**C Syntax**

```c
struct UXDDiskOptions {
    WORD structSize;
    char FAR *fileName;
};
```
Members

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>structSize</td>
<td>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.</td>
</tr>
<tr>
<td>fileName</td>
<td>Specifies the pointer to the null-terminated string that contains the file name under which you want your disk file saved.</td>
</tr>
</tbody>
</table>

Delphi Record Listing

```delphi
type
    UXDDiskOptions = record
        structSize: Word;
        fileName: PChar;
    end;

UXDMAPIOptions

UXDMAPIOptions contains e-mail information that is used by PEEnableEventInfo (page 446), when you want to export to a MAPI e-mail destination.

C Syntax

```c
struct UXDMAPIOptions {
    WORD structSize;
    char FAR *toList;
    char FAR *ccList;
    char FAR *subject;
    char FAR *message;
    WORD nRecipients;
    lpMapiRecipDesc recipients;
};
```

Members

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>structSize</td>
<td>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.</td>
</tr>
<tr>
<td>toList</td>
<td>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.</td>
</tr>
<tr>
<td>ccList</td>
<td>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.</td>
</tr>
</tbody>
</table>
UXDMAPIOptions

UXDMAPIOptions contains e-mail information that is used by the **PEEnableEventInfo** (page 446), when you want to export to a MAPI e-mail destination.

### C Syntax

```c
struct UXDMAPIOptions {
    WORD structSize;
    char FAR *toList;
    char FAR *ccList;
    char FAR *subject;
    char FAR *message;
};
```

### Members

<table>
<thead>
<tr>
<th>member</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>structSize</td>
<td>Specifies the size of the UXDMAPIOptions structure. You must initialize this</td>
</tr>
<tr>
<td></td>
<td>member to be the size of whatever it is, for example, options.structSize =</td>
</tr>
<tr>
<td></td>
<td>UXDMAPIOptionsSize.</td>
</tr>
</tbody>
</table>
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 (page 446), when you want to export to Microsoft Exchange.

C Syntax

struct UXDPostFolderOptions {
   WORD structSize;
   LPSTR pszProfile;
   LPSTR pszPassword;
   WORD wDestType;
   LPSTR pszFolderPath;
};

Members

<table>
<thead>
<tr>
<th>structSize</th>
<th>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.</th>
</tr>
</thead>
<tbody>
<tr>
<td>pszProfile</td>
<td>Specifies the Exchange profile.</td>
</tr>
<tr>
<td>pszPassword</td>
<td>Specifies the Exchange password.</td>
</tr>
</tbody>
</table>
C/VB/Delphi

wDestType Specifies the type of report to export to. Use one of the following:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UXDEExchFolderType</td>
<td>0</td>
<td>wDestType for Microsoft Exchange folder.</td>
</tr>
<tr>
<td>UXDPDocMessage</td>
<td>1009</td>
<td>wDestType for folder messages.</td>
</tr>
<tr>
<td>UXDPostPersonalReport</td>
<td>1010</td>
<td>wDestType for personal report.</td>
</tr>
<tr>
<td>UXDPostFolderReport</td>
<td>1011</td>
<td>wDestType for folder report.</td>
</tr>
</tbody>
</table>

pszFolderPath Specifies the Exchange path where you want the program to place the exported file.

Delphi Record Listing

```delphi
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>*)

**UXDVIMOOptions**

UXDVIMOOptions contains e-mail message information that is used by **PEEnableEventInfo** (page 446), when you want to export a VIM e-mail destination.

C Syntax

```c
struct UXDVIMOOptions {
    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 Seagate Crystal Reports.*

**Members**

<table>
<thead>
<tr>
<th>structSize</th>
<th>Specifies the size of the UXDVIMOOptions structure. You must initialize this member to be the size of whatever it is, for example, options.structSize = UXDVIMOOptionsSize.</th>
</tr>
</thead>
<tbody>
<tr>
<td>toList</td>
<td>Specifies the pointer to the null-terminated string that contains the “To” list to which you want your e-mail message directed.</td>
</tr>
</tbody>
</table>
UXFCharSeparatedOptions

UXFCharSeparatedOptions contains number and date information used by the PEEnableEventInfo (page 446), structure when you want to export in a Character Separated format and hard code number and/or date options.

**C Syntax**

```c
struct UXFCharSeparatedOptions {
    WORD structSize;
    BOOL useReportNumberFormat;
    BOOL useReportDateFormat;
    char stringDelimiter;
    char FAR *fieldDelimiter;
};
```

**Members**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ccList</td>
<td>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.</td>
</tr>
<tr>
<td>bccList</td>
<td>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.</td>
</tr>
<tr>
<td>subject</td>
<td>Specifies the pointer to the null-terminated string you want to appear as the subject line in the e-mail message.</td>
</tr>
<tr>
<td>message</td>
<td>Specifies the pointer to the null-terminated string you want to appear as the body of your e-mail message.</td>
</tr>
<tr>
<td>structSize</td>
<td>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.</td>
</tr>
<tr>
<td>useReportNumberFormat</td>
<td>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.</td>
</tr>
<tr>
<td>useReportDateFormat</td>
<td>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.</td>
</tr>
<tr>
<td>stringDelimiter</td>
<td>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.</td>
</tr>
</tbody>
</table>
**Delphi Record Listing**

```delphi
type
  UXFCharSeparatedOptions = record
    structSize: Word;
    useReportNumberFormat: Bool;
    useReportDateFormat: Bool;
    stringDelimiter: Char;
    fieldDelimiter: PChar;
  end;
```

**UXFCommaTabSeparatedOptions**

UXFCommaTabSeparatedOptions contains number and date information used by `PEEnableEventInfo` (page 446), when you want to export in a Comma-separated or Tab-separated format and hard code number and/or date options.

**C Syntax**

```c
struct UXFCommaTabSeparatedOptions {
  WORD structSize;
  BOOL useReportNumberFormat;
  BOOL useReportDateFormat;
};
```

**Members**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>structSize</td>
<td>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.</td>
</tr>
<tr>
<td>useReportNumberFormat</td>
<td>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.</td>
</tr>
<tr>
<td>useReportDateFormat</td>
<td>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.</td>
</tr>
</tbody>
</table>
Delphi Record Listing

```delphi
type
  UXFCommaTabSeparatedOptions = record
    structSize: Word;
    useReportNumberFormat: Bool;
    useReportDateFormat: Bool;
  end;
```

UXFDIFOptions

UXFDIFOptions contains number and date information used by `PEEnableEventInfo` (page 446), when you want to export in a DIF format (Data Interchange Format) and hard code number and/or date options.

C Syntax

```c
struct UXFDIFOptions {
  WORD structSize;
  BOOL useReportNumberFormat;
  BOOL useReportDateFormat;
};
```

Members

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>structSize</td>
<td>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.</td>
</tr>
<tr>
<td>useReportNumberFormat</td>
<td>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.</td>
</tr>
<tr>
<td>useReportDateFormat</td>
<td>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.</td>
</tr>
</tbody>
</table>

Delphi Record Listing

```delphi
type
  UXFDIFOptions = record
    structSize: Word;
    useReportNumberFormat: Bool;
    useReportDateFormat: Bool;
  end;
```
UXFHTML3Options

UXFHTML3Options contains options used by PEEnableEventInfo (page 446), when you are exporting to HTML format.

C Syntax

```c
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. |
| fileNamel | 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

```delphi
type
    UXFHTML3Options = record
        structSize: Word;
        fileName: PChar;
    end;
```

UXFODBCOptions

UXFODBCOptions contains information used by PEEnableEventInfo (page 446), whenever you export in ODBC format.

C Syntax

```c
struct UXFODBCOptions {
    WORD structSize;
    char FAR *dataSourceName;
    char FAR *dataSourceUserID;
    char FAR *dataSourcePassword;
    char FAR *exportTableName;
};
```
Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>structSize</td>
<td>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.</td>
</tr>
<tr>
<td>dataSourceName</td>
<td>Specifies the name of the data source that you want to export to.</td>
</tr>
<tr>
<td>dataSourceUserID</td>
<td>Specifies the User ID that you need to connect to the data source.</td>
</tr>
<tr>
<td>dataSourcePassword</td>
<td>Specifies the Password that you need to connect to the data source.</td>
</tr>
<tr>
<td>exportTableName</td>
<td>Specifies the name of the table you want to export to in the data source.</td>
</tr>
</tbody>
</table>

Delphi Record Listing

type
UXFODBCOptions = record
  structSize: Word;
  dataSourceName: PChar;
  dataSourceUserID: PChar;
  dataSourcePassword: PChar;
  exportTableName: PChar;
end;

UXFPaginatedTextOptions

UXFPaginatedTextOptions contains information used by PEEnableEventInfo (page 446), whenever you export in paginated text format.

C Syntax

```c
struct UXFPaginatedTextOptions {
  WORD structSize;
  WORD nLinesPerPage;
};
```

Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>structSize</td>
<td>Specifies the size of the UXFPaginatedTextOptions structure. Initialize this member to UXFPaginatedTextOptionsSize.</td>
</tr>
<tr>
<td>nLinesPerPage</td>
<td>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, Page 289, 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.</td>
</tr>
</tbody>
</table>
Delphi Record Listing

```delphi
type
    UXFPaginatedTextOptions = Record
    structSize: Word;
    nLinesPerPage: Word;
end;
```

**UXFRecordStyleOptions**

UXFRecordStyleOptions contains number and date information used by `PEEnableEventInfo` (page 446), when you want to export in a Record style (columns of values) format and hard code number and/or date options.

**C Syntax**

```c
struct UXFRecordStyleOptions {
    WORD structSize;
    BOOL useReportNumberFormat;
    BOOL useReportDateFormat;
};
```

**Members**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>structSize</td>
<td>Specifies the size of the UXFRecordStyleOptions structure. Initialize this member to UXFRecordStyleOptionsSize.</td>
</tr>
<tr>
<td>useReportNumberFormat</td>
<td>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.</td>
</tr>
<tr>
<td>useReportDateFormat</td>
<td>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.</td>
</tr>
</tbody>
</table>

Delphi Record Listing

```delphi
type
    UXFRecordStyleOptions = record
    structSize: Word;
    useReportNumberFormat: Bool;
    useReportDateFormat: Bool;
end;
```
MICROSOFT WINDOWS STRUCTURES

The following Microsoft Windows structures are discussed in this section.

COLORREF, Page 523
DEVMODE, Page 523

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;
    }
} DEVMODE;
Members

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dmDeviceName</td>
<td>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.</td>
</tr>
<tr>
<td>dmSpecVersion</td>
<td>Specifies the version number of the initialization data specification on which the structure is based.</td>
</tr>
<tr>
<td>dmDriverVersion</td>
<td>Specifies the printer driver version number assigned by the printer driver developer.</td>
</tr>
<tr>
<td>dmSize</td>
<td>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.</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMORIENT_PORTRAIT</td>
<td></td>
</tr>
<tr>
<td>DMORIENT_LANDSCAPE</td>
<td></td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMPAPER_LETTER</td>
<td>Letter, 8 1/2 by 11 inches</td>
</tr>
<tr>
<td>DMPAPER_LEGAL</td>
<td>Legal, 8 1/2 by 14 inches</td>
</tr>
<tr>
<td>DMPAPER_A4 A4</td>
<td>Sheet, 210 by 297 millimeters</td>
</tr>
<tr>
<td>DMPAPER_CSHEET</td>
<td>C Sheet, 17 by 22 inches</td>
</tr>
<tr>
<td>DMPAPER_DSHEET</td>
<td>D Sheet, 22 by 34 inches</td>
</tr>
<tr>
<td>DMPAPER_ESHEET</td>
<td>E Sheet, 34 by 44 inches</td>
</tr>
<tr>
<td>DMPAPER_LETTERSMALL</td>
<td>Letter Small, 8 1/2 by 11 inches</td>
</tr>
<tr>
<td>DMPAPER_TABLOID</td>
<td>Tabloid, 11 by 17 inches</td>
</tr>
<tr>
<td>DMPAPER_LEDGER</td>
<td>Ledger, 17 by 11 inches</td>
</tr>
<tr>
<td>DMPAPER_STATEMENT</td>
<td>Statement, 5 1/2 by 8 1/2 inches</td>
</tr>
<tr>
<td>DMPAPER_EXECUTIVE</td>
<td>Executive, 7 1/4 by 10 1/2 inches</td>
</tr>
<tr>
<td>DMPAPER_A3</td>
<td>A3 sheet, 297 by 420 millimeters</td>
</tr>
<tr>
<td>DMPAPER_A4SMALL</td>
<td>A4 small sheet, 210 by 297 millimeters</td>
</tr>
<tr>
<td>DMPAPER_A5</td>
<td>A5 sheet, 148 by 210 millimeters</td>
</tr>
<tr>
<td>DMPAPER_B4</td>
<td>B4 sheet, 250 by 354 millimeters</td>
</tr>
<tr>
<td>DMPAPER_B5</td>
<td>B5 sheet, 182-by-257-millimeter paper</td>
</tr>
<tr>
<td>DMPAPER_FOLIO</td>
<td>Folio, 8-1/2-by-13-inch paper</td>
</tr>
<tr>
<td>DMPAPER_QUARTO</td>
<td>Quarto, 215-by-275-millimeter paper</td>
</tr>
<tr>
<td>DMPAPER_10X14</td>
<td>10-by-14-inch sheet</td>
</tr>
<tr>
<td>DMPAPER_11X17</td>
<td>11-by-17-inch sheet</td>
</tr>
<tr>
<td>Constant</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>DMPAPER_NOTE</td>
<td>Note, 8 1/2 by 11 inches</td>
</tr>
<tr>
<td>DMPAPER_ENV_9</td>
<td>#9 Envelope, 3 7/8 by 8 7/8 inches</td>
</tr>
<tr>
<td>DMPAPER_ENV_10</td>
<td>#10 Envelope, 4 1/8 by 9 1/2 inches</td>
</tr>
<tr>
<td>DMPAPER_ENV_11</td>
<td>#11 Envelope, 4 1/2 by 10 3/8 inches</td>
</tr>
<tr>
<td>DMPAPER_ENV_12</td>
<td>#12 Envelope, 4 3/4 by 11 inches</td>
</tr>
<tr>
<td>DMPAPER_ENV_14</td>
<td>#14 Envelope, 5 by 11 1/2 inches</td>
</tr>
<tr>
<td>DMPAPER_ENV_DL</td>
<td>DL Envelope, 110 by 220 millimeters</td>
</tr>
<tr>
<td>DMPAPER_ENV_C5</td>
<td>C5 Envelope, 162 by 229 millimeters</td>
</tr>
<tr>
<td>DMPAPER_ENV_C3</td>
<td>C3 Envelope, 324 by 458 millimeters</td>
</tr>
<tr>
<td>DMPAPER_ENV_C4</td>
<td>C4 Envelope, 229 by 324 millimeters</td>
</tr>
<tr>
<td>DMPAPER_ENV_C6</td>
<td>C6 Envelope, 114 by 162 millimeters</td>
</tr>
<tr>
<td>DMPAPER_ENV_C65</td>
<td>C65 Envelope, 114 by 229 millimeters</td>
</tr>
<tr>
<td>DMPAPER_ENV_B4</td>
<td>B4 Envelope, 250 by 353 millimeters</td>
</tr>
<tr>
<td>DMPAPER_ENV_B5</td>
<td>B5 Envelope, 176 by 250 millimeters</td>
</tr>
<tr>
<td>DMPAPER_ENV_B6</td>
<td>B6 Envelope, 176 by 125 millimeters</td>
</tr>
<tr>
<td>DMPAPER_ENV_ITALY</td>
<td>Italy Envelope, 110 by 230 millimeters</td>
</tr>
<tr>
<td>DMPAPER_ENV_MONARCH</td>
<td>Monarch Envelope, 3 7/8 by 7 1/2 inches</td>
</tr>
<tr>
<td>DMPAPER_ENV_PERSONAL</td>
<td>6 3/4 Envelope, 3 5/8 by 6 1/2 inches</td>
</tr>
<tr>
<td>DMPAPER_FANFOLD_US</td>
<td>US Std Fanfold, 14 7/8 by 11 inches</td>
</tr>
<tr>
<td>DMPAPER_FANFOLD_STD_GERMAN</td>
<td>German Std Fanfold, 8 1/2 by 12 inches</td>
</tr>
<tr>
<td>DMPAPER_FANFOLD_LGL_GERMAN</td>
<td>German Legal Fanfold, 8 1/2 by 13 inches</td>
</tr>
</tbody>
</table>

- **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.
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dmCopies</td>
<td>Selects the number of copies printed if the device supports multiple-page copies.</td>
</tr>
<tr>
<td>dmDefaultSource</td>
<td>Reserved. This member must be set to 0.</td>
</tr>
<tr>
<td>dmPrintQuality</td>
<td>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.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMRES_HIGH</td>
<td></td>
</tr>
<tr>
<td>DMRES_MEDIUM</td>
<td></td>
</tr>
<tr>
<td>DMRES_LOW</td>
<td></td>
</tr>
<tr>
<td>DMRES_DRAFT</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dmColor</td>
<td>Switches between color and monochrome on color printers. Following are the possible values:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMCOLOR_COLOR</td>
<td></td>
</tr>
<tr>
<td>DMCOLOR_MONOCHROME</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dmDuplex</td>
<td>Selects duplex or double-sided printing for printers capable of duplex printing. Following are the possible values:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMDUP_SIMPLEX</td>
<td></td>
</tr>
<tr>
<td>DMDUP_HORIZONTAL</td>
<td></td>
</tr>
<tr>
<td>DMDUP_VERTICAL</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dmYResolution</td>
<td>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.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dmTTOption</td>
<td>Specifies how TrueType® fonts should be printed. This member can be one of the following values:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMTT_BITMAP</td>
<td>Prints TrueType fonts as graphics. This is the default action for dot-matrix printers.</td>
</tr>
<tr>
<td>DMTT_DOWNLOAD</td>
<td>Downloads TrueType fonts as soft fonts. This is the default action for Hewlett-Packard printers that use Printer Control Language (PCL).</td>
</tr>
<tr>
<td>DMTT_SUBDEV</td>
<td>Substitute device fonts for TrueType fonts. This is the default action for PostScript® printers.</td>
</tr>
</tbody>
</table>
dmCollate Specifies whether collation should be used when printing multiple copies. Using DMCOLLABATE_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 one of the following values:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMCOLLABATE_TRUE</td>
<td>Collate when printing multiple copies.</td>
</tr>
<tr>
<td>DMCOLLABATE_FALSE</td>
<td>Do NOT collate when printing multiple copies.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DM_GRAYSCALE</td>
<td>Specifies that the display is a non-color device. If this flag is not set, color is assumed.</td>
</tr>
<tr>
<td>DM_INTERLACED</td>
<td>Specifies that the display mode is interlaced. If the flag is not set, non-interlaced is assumed.</td>
</tr>
</tbody>
</table>

dmDisplay Frequency 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.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMICMMETHOD_NONE</td>
<td>Specifies that ICM is disabled.</td>
</tr>
<tr>
<td>DMICMMETHOD_SYSTEM</td>
<td>Specifies that ICM is handled by Windows.</td>
</tr>
<tr>
<td>DMICMMETHOD_DRIVER</td>
<td>Specifies that ICM is handled by the device driver.</td>
</tr>
<tr>
<td>DMICMMETHOD_DEVICE</td>
<td>Specifies that ICM is handled by the destination device.</td>
</tr>
</tbody>
</table>
### 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.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMICM_ABS_COLORIMETRIC</td>
<td>Color matching should optimize to match the exact color requested without white point mapping. This value is most appropriate for use with proofing.</td>
</tr>
<tr>
<td>DMICM_COLORMETRIC</td>
<td>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.</td>
</tr>
<tr>
<td>DMICM_CONTRAST</td>
<td>Color matching should optimize for color contrast. This value is the most appropriate choice for scanned or photographic images when dithering is desired.</td>
</tr>
<tr>
<td>DMICM_SATURATE</td>
<td>Color matching should optimize for color saturation. This value is the most appropriate choice for business graphs when dithering is not desired.</td>
</tr>
</tbody>
</table>

### 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.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMMEDIA_STANDARD</td>
<td>Plain paper.</td>
</tr>
<tr>
<td>DMMEDIA_GLOSSY</td>
<td>Glossy paper.</td>
</tr>
<tr>
<td>DMMEDIA_TRANSPARENCY</td>
<td>Transparent film.</td>
</tr>
</tbody>
</table>

### 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.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMDITHER_NONE</td>
<td>No dithering.</td>
</tr>
<tr>
<td>DMDITHER_COARSE</td>
<td>Dithering with a coarse brush.</td>
</tr>
<tr>
<td>DMDITHER_FINE</td>
<td>Dithering with a fine brush.</td>
</tr>
</tbody>
</table>
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/

<table>
<thead>
<tr>
<th>DMDITHER_LINEART</th>
<th>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.</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMDITHER_ERRORDIFFUSION</td>
<td>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.</td>
</tr>
<tr>
<td>DMReserved1</td>
<td>Windows 95/98, Windows 2000: Not used; must be zero.</td>
</tr>
<tr>
<td>DMReserved2</td>
<td>Windows 95/98, Windows 2000: Not used; must be zero.</td>
</tr>
<tr>
<td>DMWidth</td>
<td>Windows NT/Windows 2000: This member must be zero. Windows 95/98: This member is not supported.</td>
</tr>
<tr>
<td>DMHeight</td>
<td>Windows NT/Windows 2000: This member must be zero. Windows 95/98: This member is not supported.</td>
</tr>
<tr>
<td>DMReserved1</td>
<td>Windows 95/98, Windows 2000: Not used; must be zero.</td>
</tr>
<tr>
<td>DMReserved2</td>
<td>Windows 95/98, Windows 2000: Not used; must be zero.</td>
</tr>
</tbody>
</table>

DMDITHER_GRAYSCALE Device does grayscaling.
PRINT ENGINE CONSTANTS

This section includes the following CRPE constants, listed alphabetically.

Area/Section Format Formula Constants, Page 533
Chart Options Constants, Page 533
  — Chart Bar Size Constants, Page 534
  — Chart Color Constants, Page 534
  — Chart Data Point Constants, Page 534
  — Chart Gridline Constants, Page 534
  — Chart Legend Placement Constants, Page 535
  — Chart Marker Shape Constants, Page 535
  — Chart Marker Size Constants, Page 535
  — Chart Number Format Constants, Page 535
  — Chart Pie Size Constants, Page 536
  — Chart Slice Detachment Constants, Page 536
  — Chart Viewing Angle Constants, Page 536
Database Type Constants, Page 537
Error Codes, Page 537
Event Id Constants, Page 541
Field Mapping Type Constants, Page 542
Formula Syntax Constants, Page 543
Graph Subtype Constants, Page 543
  — Bar Charts, Page 543
  — Line Charts, Page 544
  — Area Charts, Page 544
  — Pie Charts, Page 544
  — Doughnut Charts, Page 544
— 3D Riser Charts, Page 545
— 3D Surface Charts, Page 545
— Scatter Charts, Page 545
— Radar Charts, Page 545
— Bubble Charts, Page 545
— Stock (High/Low/Close Type) Charts, Page 546
— Misc Chart Types, Page 546

Graph Text Font Constants, Page 546
Graph Title Type Constants, Page 546
Graph Type Constants, Page 547

Group Condition Constants, Page 547
— Group Condition Masks and Type Constants, Page 547
— All field types except Date and Boolean, Page 548
— Date and DateTime Fields, Page 548
— DateTime and Time Fields, Page 548
— Boolean Fields, Page 548

Job Destination Constants, Page 549
Job Status Constants, Page 549
Object Type Constants, Page 549
Ole Object Type Constants, Page 550
Ole Object Update Constants, Page 550

Parameter Field Value Type Constants, Page 550
Range Info Constants, Page 551
Section Codes, Page 551
Sort Method Constants, Page 551
Sort Order Constants, Page 551
Track Cursor Constants, Page 552
Zoom Level Constants, Page 553
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 have the same value as `PE_FFN_AREASECTION_VISIBILITY` and `PEP_FFN_SECTION_BACK_COLOR`. All new Seagate Crystal Reports applications should use `PE_FFN_AREASECTION_VISIBILITY` and `PEP_FFN_SECTION_BACK_COLOR`.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>PEP_FFN_AREASECTION_VISIBILITY</code></td>
<td>Area and Section format</td>
</tr>
<tr>
<td><code>PEP_FFN_SECTION_VISIBILITY</code></td>
<td>Section format</td>
</tr>
<tr>
<td><code>PEP_FFN_SHOW_AREA</code></td>
<td>Area format</td>
</tr>
<tr>
<td><code>PEP_FFN_NEW_PAGE_BEFORE</code></td>
<td>Area and Section format</td>
</tr>
<tr>
<td><code>PEP_FFN_NEW_PAGE_AFTER</code></td>
<td>Area and Section format</td>
</tr>
<tr>
<td><code>PEP_FFN_KEEP_TOGETHER</code></td>
<td>Area and Section format</td>
</tr>
<tr>
<td><code>PEP_FFN_SUPPRESS_BLANK_SECTION</code></td>
<td>Section format</td>
</tr>
<tr>
<td><code>PEP_FFN_RESET_PAGE_N_AFTER</code></td>
<td>Area and Section format</td>
</tr>
<tr>
<td><code>PEP_FFN_PRINT_AT_BOTTOM_OF_PAGE</code></td>
<td>Area and Section format</td>
</tr>
<tr>
<td><code>PEP_FFN_UNDERLAY_SECTION</code></td>
<td>Section format</td>
</tr>
<tr>
<td><code>PEP_FFN_SECTION_BACK_COLOUR</code></td>
<td>Section format</td>
</tr>
<tr>
<td><code>PEP_FFN_SECTION_BACK_COLOR</code></td>
<td>Section format</td>
</tr>
</tbody>
</table>

Chart Options Constants

The following chart options constants are supported.

- Chart Bar Size Constants, Page 534
- Chart Color Constants, Page 534
- Chart Data Point Constants, Page 534
- Chart Gridline Constants, Page 534
- Chart Legend Layout Constants, Page 534
- Chart Legend Placement Constants, Page 535
- Chart Marker Shape Constants, Page 535
- Chart Marker Size Constants, Page 535
- Chart Number Format Constants, Page 535
- Chart Pie Size Constants, Page 536
- Chart Slice Detachment Constants, Page 536
- Chart Viewing Angle Constants, Page 536
# Chart Bar Size Constants

<table>
<thead>
<tr>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_GBS_MINIMUMBARSIZE</td>
</tr>
<tr>
<td>PE_GBS_SMALLBARSIZE</td>
</tr>
<tr>
<td>PE_GBS_AVERAGEBARSIZE</td>
</tr>
<tr>
<td>PE_GBS_LARGEBARSIZE</td>
</tr>
<tr>
<td>PE_GBS_MAXIMUMBARSIZE</td>
</tr>
</tbody>
</table>

# Chart Color Constants

<table>
<thead>
<tr>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_GCR_COLORCHART</td>
</tr>
<tr>
<td>PE_GCR_BLACKANDWHITECHART</td>
</tr>
</tbody>
</table>

# Chart Data Point Constants

<table>
<thead>
<tr>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_GDP_NONE</td>
</tr>
<tr>
<td>PE_GDP_SHOWLABEL</td>
</tr>
<tr>
<td>PE_GDP_SHOWVALUE</td>
</tr>
</tbody>
</table>

# Chart Gridline Constants

<table>
<thead>
<tr>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_GGT_NOGRIDLINES</td>
</tr>
<tr>
<td>PE_GGT_MINORGRIDLINES</td>
</tr>
<tr>
<td>PE_GGT_MAJORGRIDLINES</td>
</tr>
<tr>
<td>PE_GGT_MAJORANDMINORGRIDLINES</td>
</tr>
</tbody>
</table>

# Chart Legend Layout Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_GLL_PERCENTAGE</td>
<td>Used for PEGetGraphOptionInfo only. Do not use for PESetGraphOptionInfo.</td>
</tr>
<tr>
<td>PE_GLL_AMOUNT</td>
<td></td>
</tr>
<tr>
<td>PE_GLL_CUSTOM</td>
<td></td>
</tr>
</tbody>
</table>
### Chart Legend Placement Constants

<table>
<thead>
<tr>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_GLP_PLACEUPPERRIGHT</td>
</tr>
<tr>
<td>PE_GLP_PLACEBOTTOMCENTER</td>
</tr>
<tr>
<td>PE_GLP_PLACETOPCENTER</td>
</tr>
<tr>
<td>PE_GLP_PLACERIGHT</td>
</tr>
<tr>
<td>PE_GLP_PLACELEFT</td>
</tr>
</tbody>
</table>

### Chart Marker Shape Constants

<table>
<thead>
<tr>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_GMSP_RECTANGLESHAPE</td>
</tr>
<tr>
<td>PE_GMSP_CIRCLESHAPE</td>
</tr>
<tr>
<td>PE_GMSP_DIAMONDSHAPE</td>
</tr>
<tr>
<td>PE_GMSP_TRIANGLESHAPE</td>
</tr>
</tbody>
</table>

### Chart Marker Size Constants

<table>
<thead>
<tr>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_GMS_SMALLMARKERS</td>
</tr>
<tr>
<td>PE_GMS_MEDIUSSMALLMARKERS</td>
</tr>
<tr>
<td>PE_GMS_MEDIUMLMARKERS</td>
</tr>
<tr>
<td>PE_GMS_MEDIUMLARGEMARKERS</td>
</tr>
<tr>
<td>PE_GMS_LARGEMARKERS</td>
</tr>
</tbody>
</table>

### Chart Number Format Constants

<table>
<thead>
<tr>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_GNF_NODECIMAL</td>
</tr>
<tr>
<td>PE_GNF_ONEDecimal</td>
</tr>
<tr>
<td>PE_GNF_TWODEcimal</td>
</tr>
<tr>
<td>PE_GNF_CURRENCYNODECIMAL</td>
</tr>
<tr>
<td>PE_GNF_CURRENCYTWODEcimal</td>
</tr>
<tr>
<td>PE_GNF_PERCENTNODECIMAL</td>
</tr>
<tr>
<td>PE_GNF_PERCENTONEDECIMAL</td>
</tr>
<tr>
<td>PE_GNF_PERCENTTWODEcimal</td>
</tr>
</tbody>
</table>
## Chart Pie Size Constants

<table>
<thead>
<tr>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_GPS_MINIMUMPIESIZE</td>
</tr>
<tr>
<td>PE_GPS_SMALLPIESIZE</td>
</tr>
<tr>
<td>PE_GPS_AVERAGEPIESIZE</td>
</tr>
<tr>
<td>PE_GPS_LARGEPIESIZE</td>
</tr>
<tr>
<td>PE_GPS_MAXIMUMPIESIZE</td>
</tr>
</tbody>
</table>

## Chart Slice Detachment Constants

<table>
<thead>
<tr>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_GDPS_NODETACHMENT</td>
</tr>
<tr>
<td>PE_GDPS_SMALLESTSLICE</td>
</tr>
<tr>
<td>PE_GDPS_LARGESTSLICE</td>
</tr>
</tbody>
</table>

## Chart Viewing Angle Constants

<table>
<thead>
<tr>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_GVA_STANDARDVIEW</td>
</tr>
<tr>
<td>PE_GVA_TALLVIEW</td>
</tr>
<tr>
<td>PE_GVA_TOPVIEW</td>
</tr>
<tr>
<td>PE_GVA_DISTORTEDVIEW</td>
</tr>
<tr>
<td>PE_GVA_SHORTVIEW</td>
</tr>
<tr>
<td>PE_GVA_GROUPEYEVIEW</td>
</tr>
<tr>
<td>PE_GVA_GROUPEMPHASISVIEW</td>
</tr>
<tr>
<td>PE_GVA_FEWSERIESVIEW</td>
</tr>
<tr>
<td>PE_GVA_FEWGROUPEVIEW</td>
</tr>
<tr>
<td>PE_GVA_DISTORTEDSTDVIEW</td>
</tr>
<tr>
<td>PE_GVA_SHORTERVIEW</td>
</tr>
<tr>
<td>PE_GVA_THICKSERIESVIEW</td>
</tr>
<tr>
<td>PE_GVA_THICKSTDVIEW</td>
</tr>
<tr>
<td>PE_GVA_BIRDSEYEVIEW</td>
</tr>
<tr>
<td>PE_GVA_MAXVIEW</td>
</tr>
</tbody>
</table>
### Database Type Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_DT_STANDARD</td>
<td>Standard, non-SQL databases.</td>
</tr>
<tr>
<td>PE_DT_SQL</td>
<td>SQL databases.</td>
</tr>
<tr>
<td>PE_DT_SQL_STORED_PROCEDURE</td>
<td>SQL stored procedures.</td>
</tr>
</tbody>
</table>

### Error Codes

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_ERR_NOERROR</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_NOTENOUGHMEMORY</td>
<td>500</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_INVALIDJOBNO</td>
<td>501</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_INVALIDHANDLE</td>
<td>502</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_STRINGTOOLONG</td>
<td>503</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_NOSUCHREPORT</td>
<td>504</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_NODESTINATION</td>
<td>505</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_BADFILENAMENUMBER</td>
<td>506</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_BADFILENAME</td>
<td>507</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_BADFIELDNUMBER</td>
<td>508</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_BADFIELDNAME</td>
<td>509</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_BADFORMULANAME</td>
<td>510</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_BADSORTDIRECTION</td>
<td>511</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_ENGINENOTOPEN</td>
<td>512</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_INVALIDPRINTER</td>
<td>513</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_PRINTFILEEXISTS</td>
<td>514</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_BADFORMULATEXT</td>
<td>515</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_BADGROUPSECTION</td>
<td>516</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_ENGINEBUSY</td>
<td>517</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_BADSECTION</td>
<td>518</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_NOPRINTWINDOW</td>
<td>519</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_JOBALREADYSTARTED</td>
<td>520</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_BADSUMMARYFIELD</td>
<td>521</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_NOTENOUGHSYSSRES</td>
<td>522</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_BADGROUPCONDITION</td>
<td>523</td>
<td></td>
</tr>
<tr>
<td>Constant</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>PE_ERR_JOBBUSY</td>
<td>524</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_BADREPORTFILE</td>
<td>525</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_NODEFAULTPRINTER</td>
<td>526</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_SQLSERVERERROR</td>
<td>527</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_BADLINENUMBER</td>
<td>528</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_DISKFULL</td>
<td>529</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_FILEERROR</td>
<td>530</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_INCORRECTPASSWORD</td>
<td>531</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_BADDATABASEDLL</td>
<td>532</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_BADDATABASEFILE</td>
<td>533</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_ERRORINDATABASEDLL</td>
<td>534</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_DATABASESESSION</td>
<td>535</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_DATABASELOGON</td>
<td>536</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_DATABASELOCATION</td>
<td>537</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_BADSTRUCTSIZE</td>
<td>538</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_BADDATE</td>
<td>539</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_BADEXPORTDLL</td>
<td>540</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_ERRORINEXPORTDLL</td>
<td>541</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_PREVATFIRSTPAGE</td>
<td>542</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_NEXATLASTPAGE</td>
<td>543</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_CANNOTACCESSREPORT</td>
<td>544</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_USERCANCELED</td>
<td>545</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_OLE2NOTLOADED</td>
<td>546</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_BADCROSSTABGROUP</td>
<td>547</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_NOCTSUMMARIZEDFIELD</td>
<td>548</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_DESTINATIONNOTEXPORT</td>
<td>549</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_INVALIDPAGENUMBER</td>
<td>550</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_NOTSTOREDPROCEDURE</td>
<td>552</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_INVALIDPARAMETER</td>
<td>553</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_NOTFOUND</td>
<td>554</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_INVALIDGRAPHTYPE</td>
<td>555</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_INVALIDGRAPHDATA</td>
<td>556</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_CANNOTMOVEGRAPH</td>
<td>557</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_INVALIDGRAPHTEXT</td>
<td>558</td>
<td></td>
</tr>
<tr>
<td>Constant</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>PE_ERR_INVALIDGRAPHHOPT</td>
<td>559</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_BADSECTIONHEIGHT</td>
<td>560</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_BADVALUETYPE</td>
<td>561</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_INVALIDSUBREPORTNAME</td>
<td>562</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_NOPARENTWINDOW</td>
<td>564</td>
<td>Dialog parent window</td>
</tr>
<tr>
<td>PE_ERR_INVALIDZOOMFACTOR</td>
<td>565</td>
<td>Zoom factor</td>
</tr>
<tr>
<td>PE_ERR_PAGESIZEOVERFLOW</td>
<td>567</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_LOWSYSTEMRESOURCES</td>
<td>568</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_BADGROUPNUMBER</td>
<td>570</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_INVALIDOBJECTFORMATNAME</td>
<td>571</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_INVALIDNEGATIVEVALUE</td>
<td>572</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_INVALIDMEMORYPOINTER</td>
<td>573</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_INVALIDOBJECTTYPE</td>
<td>574</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_INVALIDGRAPHDATATYPE</td>
<td>577</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_INVALIDSUBREPORTLINKNUMBER</td>
<td>582</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_SUBREPORTLINKEXIST</td>
<td>583</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_BADROWCOLVALUE</td>
<td>584</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_INVALIDSUMMARYNUMBER</td>
<td>585</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_INVALIDGRAPHDATAFIELDNUMBER</td>
<td>586</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_INVALIDSUBREPORTNUMBER</td>
<td>587</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_INVALIDFIELDSCOPE</td>
<td>588</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_FIELDINUSE</td>
<td>590</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_INVALIDPARAMETERNUMBER</td>
<td>594</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_INVALIDPAGEMARGINS</td>
<td>595</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_REPORTONSECUREQUERY</td>
<td>596</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_CANNOTOPENSECUREQUERY</td>
<td>597</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_INVALIDSECTIONNUMBER</td>
<td>598</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_SQLSERVERNOTOPENED</td>
<td>599</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_TABLENAMEEXIST</td>
<td>606</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_INVALIDCURSOR</td>
<td>607</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_FIRSTPASSNOTFINISHED</td>
<td>608</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_CREATEDDATASOURCE</td>
<td>609</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_CREATEDRILLDOWNPARAMETERS</td>
<td>610</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_CHECKFORDATASOURCECHANGES</td>
<td>613</td>
<td></td>
</tr>
<tr>
<td>Constant</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PE_ERR_STARTBACKGROUNDPROCESSING</td>
<td>614</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_SQLSERVERINUSE</td>
<td>619</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_GROUPSORTFIELDNOTSET</td>
<td>620</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_CANNOTSETSAVESUMMARIES</td>
<td>621</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_LOADOLAPDATABASEMANAGER</td>
<td>622</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_OPENOLAPCUBE</td>
<td>623</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_READOLAPCUBEDATA</td>
<td>624</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_CANNOTSAVEQUERY</td>
<td>626</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_CANNOTREADQUERYDATA</td>
<td>627</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_MAINREPORTFIELDLINKED</td>
<td>629</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_INVALIDMAPPINGTYPEVALUE</td>
<td>630</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_HITTESTFAILED</td>
<td>636</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_BADSQLEXPRESSIONNAME</td>
<td>637</td>
<td>No SQL expression by the specified <em>name</em> exists in this report.</td>
</tr>
<tr>
<td>PE_ERR_BADSQLEXPRESSIONNUMBER</td>
<td>638</td>
<td>No SQL expression by the specified <em>number</em> exists in this report.</td>
</tr>
<tr>
<td>PE_ERR_BADSQLEXPRESSIONTEXT</td>
<td>639</td>
<td>Not a valid SQL expression</td>
</tr>
<tr>
<td>PE_ERR_INVALIDDEFAULTVALUEINDEX</td>
<td>641</td>
<td>Invalid index for default value of a parameter.</td>
</tr>
<tr>
<td>PE_ERR_NOMINMAXVALUE</td>
<td>642</td>
<td>The specified PE_PF_* type does not have min/max values.</td>
</tr>
<tr>
<td>PE_ERR_INCONSISTANTTYPES</td>
<td>643</td>
<td>If both min and max values are specified in PESetParameterMinMaxValue, the value types for the min and max must be the same.</td>
</tr>
<tr>
<td>PE_ERR_CANNOTLINKTABLES</td>
<td>645</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_CREATEROOTER</td>
<td>646</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_INVALIDFIELDINDEX</td>
<td>647</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_INVALIDGRAPHTTITLETYPE</td>
<td>648</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_INVALIDGRAPHTTITLEFONTTYPE</td>
<td>649</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_PARAMTYPEDIFFERENT</td>
<td>650</td>
<td>The type used in a add/set value API for a parameter differs with it's existing type.</td>
</tr>
<tr>
<td>PE_ERR_INCONSISTANTRANGETYPES</td>
<td>651</td>
<td>The value type for both start &amp; end range values must be the same.</td>
</tr>
</tbody>
</table>
### Event Id Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_CLOSE_PRINT_WINDOW_EVENT</td>
<td></td>
</tr>
<tr>
<td>PE_ACTIVATE_PRINT_WINDOW_EVENT</td>
<td></td>
</tr>
<tr>
<td>PE_DEACTIVATE_PRINT_WINDOW_EVENT</td>
<td></td>
</tr>
<tr>
<td>PE_PRINT_BUTTON_CLICKED_EVENT</td>
<td></td>
</tr>
<tr>
<td>PE_EXPORT_BUTTON_CLICKED_EVENT</td>
<td></td>
</tr>
</tbody>
</table>

### Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_ERR_RANGEORDISCRETE</td>
<td>652</td>
<td>An operation was attempted on a discrete parameter that is only legal for range parameters or vice versa.</td>
</tr>
<tr>
<td>PE_ERR_NOTMAINREPORT</td>
<td>654</td>
<td>An operation was attempted that is disallowed for subreports.</td>
</tr>
<tr>
<td>PE_ERR_INVALIDCURRENTVALUEINDEX</td>
<td>655</td>
<td>Invalid index for current value of a parameter.</td>
</tr>
<tr>
<td>PE_ERR_LINKEDPARAMVALUE</td>
<td>656</td>
<td>Operation illegal on linked parameter.</td>
</tr>
<tr>
<td>PE_ERR_INVALIDPARAMETERRANGEINFO</td>
<td>672</td>
<td>Invalid PE_RI_XXX combination.</td>
</tr>
<tr>
<td>PE_ERR_INVALIDSORTMETHODINDEX</td>
<td>674</td>
<td>Invalid sort method index.</td>
</tr>
<tr>
<td>PE_ERR_INVALIDGRAPHSUBTYPE</td>
<td>675</td>
<td>Invalid PE_GST_XXX or PE_GST_XXX does not match PE_GT_XXX or PE_GST_XXX current graph type.</td>
</tr>
<tr>
<td>PE_ERR_BADGRAPHOPTIONINFO</td>
<td>676</td>
<td>One of the members of PEGraphOptionInfo is out of range.</td>
</tr>
<tr>
<td>PE_ERR_BADGRAPHAXISINFO</td>
<td>677</td>
<td>One of the members of PEGraphAxisInfo is out of range.</td>
</tr>
<tr>
<td>PE_ERR_INVALIDPARAMETERVALUE</td>
<td>687</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_INVALIDFORMULASYNTAXTYPE</td>
<td>688</td>
<td>Specified formula syntax not in PE_FST_XXX</td>
</tr>
<tr>
<td>PE_ERR_INVALIDCROPVALUE</td>
<td>689</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_INVALIDCOLLATIONVALUE</td>
<td>690</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_STARTPAGEGREATERSTOPPAGE</td>
<td>691</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_OTHERERROR</td>
<td>997</td>
<td></td>
</tr>
<tr>
<td>PE_ERR_INTERNALERROR</td>
<td>998</td>
<td>Programming error</td>
</tr>
<tr>
<td>PE_ERR_NOTIMPLEMENTED</td>
<td>999</td>
<td></td>
</tr>
</tbody>
</table>
### Field Mapping Type Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_FM_AUTO_FLD_MAP</td>
<td>Automatic field name mapping.</td>
</tr>
<tr>
<td>PE_FM_CRPE_PROMPT_FLD_MAP</td>
<td>CRPE provides dialog box to map field manually.</td>
</tr>
<tr>
<td>PE_FM_EVENT_DEFINED_FLD_MAP</td>
<td>CRPE provides list of fields in the report and the new database. This constant is not available in global32.bas.</td>
</tr>
</tbody>
</table>

### PE_ZOOM_LEVEL_CHANGING_EVENT Description

Include drill on graph.
Formula Syntax Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_FST_CRYSTAL</td>
<td>See Remarks below.</td>
</tr>
<tr>
<td>PE_FST_BASIC</td>
<td>See Remarks below.</td>
</tr>
</tbody>
</table>

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, Page 543
Line Charts, Page 544
Area Charts, Page 544
Pie Charts, Page 544
Doughnut Charts, Page 544
3D Riser Charts, Page 545
3D Surface Charts, Page 545
Scatter Charts, Page 545
Radar Charts, Page 545
Bubble Charts, Page 545
Stock (High/Low/Close Type) Charts, Page 546
Misc Chart Types, Page 546

Bar Charts

<table>
<thead>
<tr>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_GST_SIDEBYSIDEBARCHART</td>
</tr>
<tr>
<td>PE_GST_STACKEDBARCHART</td>
</tr>
<tr>
<td>PE_GST_PERCENTBARCHART</td>
</tr>
<tr>
<td>PE_GST_FAKED3DSIDEBYSIDEBARCHART</td>
</tr>
<tr>
<td>PE_GST_FAKED3DSTACKEDBARCHART</td>
</tr>
<tr>
<td>PE_GST_FAKED3DPERCENTBARCHART</td>
</tr>
</tbody>
</table>
### Line Charts

<table>
<thead>
<tr>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_GST_REGULARLINECHART</td>
</tr>
<tr>
<td>PE_GST_STACKEDLINECHART</td>
</tr>
<tr>
<td>PE_GST_PERCENTAGELINECHART</td>
</tr>
<tr>
<td>PE_GST_LINECHARTWITHMARKERS</td>
</tr>
<tr>
<td>PE_GST_STACKEDLINECHARTWITHMARKERS</td>
</tr>
<tr>
<td>PE_GST_PERCENTAGELINECHARTWITHMARKERS</td>
</tr>
</tbody>
</table>

### Area Charts

<table>
<thead>
<tr>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_GST_ABSOLUTEAREACHART</td>
</tr>
<tr>
<td>PE_GST_STACKEDAREACHART</td>
</tr>
<tr>
<td>PE_GST_PERCENTAREACHART</td>
</tr>
<tr>
<td>PE_GST_FAKED3DABSOLUTEAREACHART</td>
</tr>
<tr>
<td>PE_GST_FAKED3DSTACKEDAREACHART</td>
</tr>
<tr>
<td>PE_GST_FAKED3DPERCENTAREACHART</td>
</tr>
</tbody>
</table>

### Pie Charts

<table>
<thead>
<tr>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_GST_REGULARPIECHART</td>
</tr>
<tr>
<td>PE_GST_FAKED3DREGULARPIECHART</td>
</tr>
<tr>
<td>PE_GST_MULTIPLEPIECHART</td>
</tr>
<tr>
<td>PE_GST_MULTIPLEPROPORTIONALPIECHART</td>
</tr>
</tbody>
</table>

### Doughnut Charts

<table>
<thead>
<tr>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_GST_REGULARDOUGHNUTCHART</td>
</tr>
<tr>
<td>PE_GST_MULTIPLEDOUGHNUTCHART</td>
</tr>
<tr>
<td>PE_GST_MULTIPLEPROPORTIONALDOUGHNUTCHART</td>
</tr>
</tbody>
</table>
### 3D Riser Charts

<table>
<thead>
<tr>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_GST_THREEDREGULARCHART</td>
</tr>
<tr>
<td>PE_GST_THREEDPYRAMIDCHART</td>
</tr>
<tr>
<td>PE_GST_THREEDOCTAGONCHART</td>
</tr>
<tr>
<td>PE_GST_THREEDCUTCORNERSCHART</td>
</tr>
</tbody>
</table>

### 3D Surface Charts

<table>
<thead>
<tr>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_GST_THREEDSURFACEREGULARCHART</td>
</tr>
<tr>
<td>PE_GST_THREEDSURFACEWITHSIDESCHART</td>
</tr>
<tr>
<td>PE_GST_THREEDSURFACETWOHEXAGONCHART</td>
</tr>
</tbody>
</table>

### Scatter Charts

<table>
<thead>
<tr>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_GST_XYSCATTERCHART</td>
</tr>
<tr>
<td>PE_GST_XYSCATTERDUALAXISCHART</td>
</tr>
<tr>
<td>PE_GST_XYSCATTERWITHLABELSCHART</td>
</tr>
<tr>
<td>PE_GST_XYSCATTERDUALAXISWITHLABELSCHART</td>
</tr>
</tbody>
</table>

### Radar Charts

<table>
<thead>
<tr>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_GST_REGULARRADARCHART</td>
</tr>
<tr>
<td>PE_GST_STACKEDRADARCHART</td>
</tr>
<tr>
<td>PE_GST_RADARDUALAXISCHART</td>
</tr>
</tbody>
</table>

### Bubble Charts

<table>
<thead>
<tr>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_GST_REGULARBUBBLECHART</td>
</tr>
<tr>
<td>PE_GST_DUALAXISBUBBLECHART</td>
</tr>
</tbody>
</table>
Stock (High/Low/Close Type) Charts

<table>
<thead>
<tr>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_GST_HIGHLOWCHART</td>
</tr>
<tr>
<td>PE_GST_HIGHLOWOPENCLOSECHART</td>
</tr>
</tbody>
</table>

Misc Chart Types

<table>
<thead>
<tr>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_GST_UNKNOWNSUBTYPECHART</td>
</tr>
</tbody>
</table>

Graph Text Font Constants

<table>
<thead>
<tr>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_GTF_TITLEFONT</td>
</tr>
<tr>
<td>PE_GTF_SUBTITLEFONT</td>
</tr>
<tr>
<td>PE_GTF_FOOTNOTEFONT</td>
</tr>
<tr>
<td>PE_GTF_GROUPSTITLEFONT</td>
</tr>
<tr>
<td>PE_GTF_DATATITLEFONT</td>
</tr>
<tr>
<td>PE_GTF_LEGENDFONT</td>
</tr>
<tr>
<td>PE_GTF_GROUPLABELSFONT</td>
</tr>
<tr>
<td>PE_GTF_DATALABELSFONT</td>
</tr>
</tbody>
</table>

Graph Title Type Constants

<table>
<thead>
<tr>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_GTT_TITLE</td>
</tr>
<tr>
<td>PE_GTT_SUBTITLE</td>
</tr>
<tr>
<td>PE_GTT_FOOTNOTE</td>
</tr>
<tr>
<td>PE_GTT_SERIESTITLE</td>
</tr>
<tr>
<td>PE_GTT_GROUPSTITLE</td>
</tr>
<tr>
<td>PE_GTT_XAXISTITLE</td>
</tr>
<tr>
<td>PE_GTT_YAXISTITLE</td>
</tr>
<tr>
<td>PE_GTT_ZAXISTITLE</td>
</tr>
</tbody>
</table>
## Graph Type Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_GT_BARCHART</td>
<td></td>
</tr>
<tr>
<td>PE_GT_LINECHART</td>
<td></td>
</tr>
<tr>
<td>PE_GT_AREACHART</td>
<td></td>
</tr>
<tr>
<td>PE_GT_PIECHART</td>
<td></td>
</tr>
<tr>
<td>PE_GT_DOUGHNUTCHART</td>
<td></td>
</tr>
<tr>
<td>PE_GT_THREEDRISERCHART</td>
<td></td>
</tr>
<tr>
<td>PE_GT_THREEDSURFACECHART</td>
<td></td>
</tr>
<tr>
<td>PE_GT_SCATTERCHART</td>
<td></td>
</tr>
<tr>
<td>PE_GT_RADARCHART</td>
<td></td>
</tr>
<tr>
<td>PE_GT_BUBBLECHART</td>
<td></td>
</tr>
<tr>
<td>PE_GT_STOCKCHART</td>
<td></td>
</tr>
<tr>
<td>PE_GT_USERDEFINEDCHART</td>
<td>Use for PEGraphTypeInfo only.</td>
</tr>
<tr>
<td></td>
<td>Do not use for PESetGraphTypeInfo.</td>
</tr>
<tr>
<td>PE_GT_UNKNOWNTYPECHART</td>
<td>Use for PEGraphTypeInfo only.</td>
</tr>
<tr>
<td></td>
<td>Do not use for PESetGraphTypeInfo.</td>
</tr>
</tbody>
</table>

## Group Condition Constants

The following tables list the Group Condition masks and constants applicable to various field types.

Group Condition Masks and Type Constants, Page 547
All field types except Date and Boolean, Page 548
Date and DateTime Fields, Page 548
DateTime and Time Fields, Page 548
Boolean Fields, Page 548

### Group Condition Masks and Type Constants

<table>
<thead>
<tr>
<th>Mask and Type Constants</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_GC_CONDITIONMASK</td>
<td>0x00FF</td>
</tr>
<tr>
<td>PE_GC_TYPEMASK</td>
<td>0x0F00</td>
</tr>
<tr>
<td>PE_GC_TYPEOTHER</td>
<td>0x0000</td>
</tr>
<tr>
<td>PE_GC_TYPEDATE</td>
<td>0x0200</td>
</tr>
<tr>
<td>PE_GC_TYPEBOOLEAN</td>
<td>0x0400</td>
</tr>
<tr>
<td>PE_GC_TYPETIME</td>
<td>0x0800</td>
</tr>
</tbody>
</table>
### All field types except Date and Boolean

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_GC_ANYCHANGE</td>
<td>Triggers a grouping every time there is a change.</td>
</tr>
</tbody>
</table>

### Date and DateTime Fields

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_GC_DAILY</td>
<td>Triggers a grouping every time the date changes.</td>
</tr>
<tr>
<td>PE_GC_WEEKLY</td>
<td>Triggers a grouping every time the date changes from one week to the next.</td>
</tr>
<tr>
<td>PE_GC_BIWeekly</td>
<td>Triggers a grouping every time the date changes from one two-week period to the next.</td>
</tr>
<tr>
<td>PE_GC_SEMI_MONTHY</td>
<td>Triggers a grouping every time the date changes from one half-month period to the next.</td>
</tr>
<tr>
<td>PE_GC_MONTHLY</td>
<td>Triggers a grouping every time the date changes from one month to the next.</td>
</tr>
<tr>
<td>PE_GC_QUARTERLY</td>
<td>Triggers a grouping every time the date changes from one calendar quarter to the next.</td>
</tr>
<tr>
<td>PE_GC_SEMI_ANNUALLY</td>
<td>Triggers a grouping every time the date changes from one half-year period to the next.</td>
</tr>
<tr>
<td>PE_GC_ANNUALLY</td>
<td>Triggers a grouping every time the date changes from one year to the next.</td>
</tr>
</tbody>
</table>

### Date/Time and Time Fields

<table>
<thead>
<tr>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_GC_BYSECOND</td>
</tr>
<tr>
<td>PE_GC_BYMINUTE</td>
</tr>
<tr>
<td>PE_GC_BYHOUR</td>
</tr>
<tr>
<td>PE_GC_BAAMPM</td>
</tr>
</tbody>
</table>

### Boolean Fields

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_GC_TOYES</td>
<td>Triggers a grouping every time the sort- and group-by field value changes from No to Yes.</td>
</tr>
<tr>
<td>PE_GC_TONO</td>
<td>Triggers a grouping every time the sort- and group-by field value changes from Yes to No.</td>
</tr>
</tbody>
</table>
### Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_GC_EVERYYES</td>
<td>Triggers a grouping every time the sort- and group-by field value is Yes.</td>
</tr>
<tr>
<td>PE_GC_EVERYNO</td>
<td>Triggers a grouping every time the sort- and group-by field value is No.</td>
</tr>
<tr>
<td>PE_GC_NEXTISYES</td>
<td>Triggers a grouping every time the next value in the sort- and group-by field is Yes.</td>
</tr>
<tr>
<td>PE_GC_NEXTISNO</td>
<td>Triggers a grouping every time the next value in the sort- and group-by field is No.</td>
</tr>
</tbody>
</table>

#### Job Destination Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_TO_NOWHERE</td>
<td>No destination.</td>
</tr>
<tr>
<td>PE_TO_WINDOW</td>
<td>Print to window.</td>
</tr>
<tr>
<td>PE_TO_PRINTER</td>
<td>Print to printer.</td>
</tr>
<tr>
<td>PE_TO_EXPORT</td>
<td>Export.</td>
</tr>
<tr>
<td>PE_FROM_QUERY</td>
<td>From a query.</td>
</tr>
</tbody>
</table>

#### Job Status Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_JOBNOTSTARTED</td>
<td></td>
</tr>
<tr>
<td>PE_JOBINPROGRESS</td>
<td></td>
</tr>
<tr>
<td>PE_JOBCOMPLETED</td>
<td></td>
</tr>
<tr>
<td>PE_JOBFAILED</td>
<td>An error occurred.</td>
</tr>
<tr>
<td>PE_JOBCANCELLED</td>
<td>The job was canceled by user.</td>
</tr>
<tr>
<td>PE_JOBHALTED</td>
<td>The job was halted because of too many records or too much time.</td>
</tr>
</tbody>
</table>

#### Object Type Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_OI_FIELDOBJECT</td>
<td></td>
</tr>
<tr>
<td>PE_OI_TEXTOBJECT</td>
<td></td>
</tr>
<tr>
<td>PE_OI_LINEOBJECT</td>
<td></td>
</tr>
<tr>
<td>PE_OI_BOXOBJECT</td>
<td></td>
</tr>
<tr>
<td>PE_OI_SUBREPORTOBJECT</td>
<td></td>
</tr>
</tbody>
</table>
### Ole Object Type Constants

<table>
<thead>
<tr>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_OI_OLEOBJECT</td>
</tr>
<tr>
<td>PE_OI_GRAPHOBJECT</td>
</tr>
<tr>
<td>PE_OI_CROSSTABOBJECT</td>
</tr>
<tr>
<td>PE_OI_BLOBFIELDOBJECT</td>
</tr>
<tr>
<td>PE_OI_MAPOBJECT</td>
</tr>
<tr>
<td>PE_OI_OLAPGRIDOBJECT</td>
</tr>
</tbody>
</table>

### Ole Object Update Constants

<table>
<thead>
<tr>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_OOI_LINKEDOBJECT</td>
</tr>
<tr>
<td>PE_OOI_EMBEDDEDOBJECT</td>
</tr>
<tr>
<td>PE_OOI_STATICOBJECT</td>
</tr>
</tbody>
</table>

### Parameter Field Value Type Constants

<table>
<thead>
<tr>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_PF_NUMBER</td>
</tr>
<tr>
<td>PE_PF_CURRENCY</td>
</tr>
<tr>
<td>PE_PF_BOOLEAN</td>
</tr>
<tr>
<td>PE_PF_DATE</td>
</tr>
<tr>
<td>PE_PF_STRING</td>
</tr>
<tr>
<td>PE_PF_DATETIME</td>
</tr>
<tr>
<td>PE_PF_TIME</td>
</tr>
</tbody>
</table>
Range Info Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_RI_INCLUDEUPPERBOUND</td>
<td>1</td>
</tr>
<tr>
<td>PE_RI_INCLUDELOWERBOUND</td>
<td>2</td>
</tr>
<tr>
<td>PE_RI_NOUPPERBOUND</td>
<td>4</td>
</tr>
<tr>
<td>PE_RI_NOLOWERBOUND</td>
<td>8</td>
</tr>
</tbody>
</table>

Section Codes

<table>
<thead>
<tr>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_ALLSECTIONS</td>
</tr>
<tr>
<td>PE_SECT_PAGE_HEADER</td>
</tr>
<tr>
<td>PE_SECT_PAGE_FOOTER</td>
</tr>
<tr>
<td>PE_SECT_REPORT_HEADER</td>
</tr>
<tr>
<td>PE_SECT_REPORT_FOOTER</td>
</tr>
<tr>
<td>PE_SECT_GROUP_HEADER</td>
</tr>
<tr>
<td>PE_SECT_GROUP_FOOTER</td>
</tr>
<tr>
<td>PE_SECT_DETAIL</td>
</tr>
</tbody>
</table>

Sort Method Constants

<table>
<thead>
<tr>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_OR_NO_SORT</td>
</tr>
<tr>
<td>PE_OR_ALPHANUMERIC_ASCENDING</td>
</tr>
<tr>
<td>PE_OR_ALPHANUMERIC_DESCENDING</td>
</tr>
<tr>
<td>PE_OR_NUMERIC_ASCENDING</td>
</tr>
<tr>
<td>PE_OR_NUMERIC_DESCENDING</td>
</tr>
</tbody>
</table>

Sort Order Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_SF_DESCENDING</td>
<td>Sorts data in descending order (Z to A, 9 to 1).</td>
</tr>
<tr>
<td>PE_SF_ASCENDING</td>
<td>Sorts data in ascending order (A to Z, 1 to 9).</td>
</tr>
<tr>
<td>PE_SFORIGINAL</td>
<td>Group condition only: Sorts data in its original order.</td>
</tr>
<tr>
<td>PE_SF_SPECIFIED</td>
<td>Group condition only: Sorts data in a specified order. Read only.</td>
</tr>
</tbody>
</table>
## Track Cursor Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_TC_DEFAULT_CURSOR</td>
<td>CRPE default cursor = PE_TC_ARROW_CURSOR.</td>
</tr>
<tr>
<td>PE_TC_ARROW_CURSOR</td>
<td>Arrow cursor.</td>
</tr>
<tr>
<td>PE_TC_CROSS_CURSOR</td>
<td>Cross cursor.</td>
</tr>
<tr>
<td>PE_TC_IBEAM_CURSOR</td>
<td>I-beam cursor.</td>
</tr>
<tr>
<td>PE_TC_UPARROW_CURSOR</td>
<td>Arrow cursor pointing to the top of the screen.</td>
</tr>
<tr>
<td>PE_TC_SIZEALL_CURSOR</td>
<td>32-bit only.</td>
</tr>
<tr>
<td>PE_TC_SIZENWSE_CURSOR</td>
<td>Sizing cursor when resizing from the top, left-hand side of the screen to the bottom, right-hand side of the screen.</td>
</tr>
<tr>
<td>PE_TC_SIZENESW_CURSOR</td>
<td>Sizing cursor when resizing from the top, right-hand side of the screen to the bottom, left-hand side of the screen.</td>
</tr>
<tr>
<td>PE_TC_SIZEWE_CURSOR</td>
<td>Sizing cursor when resizing from the left side of the screen to the right side of the screen.</td>
</tr>
<tr>
<td>PE_TC_SIZENS_CURSOR</td>
<td>Sizing cursor when resizing from the top of the screen to the bottom of the screen.</td>
</tr>
<tr>
<td>PE_TC_NO_CURSOR</td>
<td>32-bit only.</td>
</tr>
<tr>
<td>PE_TC_WAIT_CURSOR</td>
<td>Wait (i.e., hourglass) cursor.</td>
</tr>
<tr>
<td>PE_TC_APPSTARTING_CURSOR</td>
<td>32-bit only.</td>
</tr>
<tr>
<td>PE_TC_HELP_CURSOR</td>
<td>32-bit only.</td>
</tr>
<tr>
<td>PE_TC_SIZE_CURSOR</td>
<td>Not used in 32-bit applications.</td>
</tr>
<tr>
<td>PE_TC_ICON_CURSOR</td>
<td>Use PE_TC_SIZEALL_CURSOR.</td>
</tr>
<tr>
<td>PE_TC_ICON_CURSOR</td>
<td>Not used in 32-bit applications.</td>
</tr>
<tr>
<td>PE_TC_ICON_CURSOR</td>
<td>Use PE_TC_ARROW_CURSOR.</td>
</tr>
<tr>
<td>PE_TC/background_PROCESS_CURSOR</td>
<td>CRPE specific cursor.</td>
</tr>
<tr>
<td>PE_TC_GRAB_HAND_CURSOR</td>
<td>CRPE specific cursor.</td>
</tr>
<tr>
<td>PE_TC_ZOOM_IN_CURSOR</td>
<td>CRPE specific cursor.</td>
</tr>
<tr>
<td>PE_TC_REPORT_SECTION_CURSOR</td>
<td>CRPE specific cursor.</td>
</tr>
<tr>
<td>PE_TC_HAND_CURSOR</td>
<td>CRPE specific cursor.</td>
</tr>
<tr>
<td>PE_TC_MAGNIFY_CURSOR</td>
<td>CRPE specific cursor. Magnifying glass cursor (used for drill-down).</td>
</tr>
</tbody>
</table>
Zoom Level Constants

<table>
<thead>
<tr>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_ZOOM_FULL_SIZE</td>
</tr>
<tr>
<td>PE_ZOOM_SIZE_FIT_ONE_SIDE</td>
</tr>
<tr>
<td>PE_ZOOM_SIZE_FIT_BOTH_SIDES</td>
</tr>
</tbody>
</table>

**Obsoleted 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**

<table>
<thead>
<tr>
<th>Obsolete Function</th>
<th>Replacement Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>PEGetGraphData</td>
<td></td>
</tr>
<tr>
<td>PEGetGraphOptions</td>
<td>PEGetGraphOptionInfo, Page 295</td>
</tr>
<tr>
<td>PEGetGraphText</td>
<td>PEGetGraphTextInfo, Page 297</td>
</tr>
<tr>
<td>PEGetGraphType</td>
<td>PEGetGraphTypeInfo, Page 298</td>
</tr>
<tr>
<td>PEGetMinimumSectionHeight</td>
<td>PEGetSectionHeight, Page 346</td>
</tr>
<tr>
<td>PEGetNParams</td>
<td>PEGetNParameterFields, Page 313</td>
</tr>
<tr>
<td>PEGetNthParam</td>
<td>PEGetNthParameterInfo, Page 324</td>
</tr>
<tr>
<td>PEGetNthParamInfo</td>
<td>PEGetParameterValueInfo, Page 338</td>
</tr>
<tr>
<td>PESetGraphData</td>
<td></td>
</tr>
<tr>
<td>PESetGraphOptions</td>
<td>PESetGroupOptions, Page 402</td>
</tr>
<tr>
<td>PESetGraphText</td>
<td>PESetGraphTextInfo, Page 399</td>
</tr>
<tr>
<td>PESetGraphType</td>
<td>PESetGraphTypeInfo, Page 400</td>
</tr>
<tr>
<td>PESetMinimumSectionHeight</td>
<td>PESetSectionHeight, Page 428</td>
</tr>
<tr>
<td>PESetNthParam</td>
<td>PESetNthParameterField, Page 409</td>
</tr>
</tbody>
</table>
### Obsolete Structures

<table>
<thead>
<tr>
<th>Obsolete Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>PECharSepFileOptions</td>
</tr>
<tr>
<td>PEGraphDataInfo</td>
</tr>
<tr>
<td>PEGraphOptions</td>
</tr>
<tr>
<td>PEGraphTextInfo</td>
</tr>
<tr>
<td>PEPParameterInfo</td>
</tr>
<tr>
<td>PEPrintFileOptions</td>
</tr>
</tbody>
</table>

### Obsolete Constants

<table>
<thead>
<tr>
<th>Obsolete Constants</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_GRAPH_XXX Graph Direction Constants</td>
</tr>
<tr>
<td>PE_SIDE_ / PE_FAKED_ / etc. Graph Type Constants</td>
</tr>
</tbody>
</table>
5

Using Crystal Report Viewers in Applications

What you will find in this chapter . . .

Application Development with Crystal Report Viewers, Page 556
...an introduction.

Seagate Crystal Report Viewer for ActiveX, Page 556
...including an introduction to programming with the Report Viewer Object Model, and adding and using the Report Viewer Object.

The Crystal Report Viewer Java Bean, Page 563
...including an introduction, comments regarding adding the Report Viewer Bean to your project and creating a simple applet with Report Viewer.
APPLICATION DEVELOPMENT WITH CRYSTAL REPORT VIEWERS

Developing applications that display reports on screen is now a straightforward process. Seagate Crystal Reports includes the Seagate 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.

Seagate 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, Seagate 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 Seagate Crystal Reports development technologies. If you are interested in creating web sites that display reports, you should also review Seagate Crystal Reports Web Administrator’s Guide, which describes the HTML and Java applet versions of the Report Viewer as well.

SEAGATE CRYSTAL REPORT VIEWER FOR ACTIVEX

The Seagate 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 Seagate 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.
The following topics are discussed in this section.

Adding the Report Viewer to a Visual Basic project, Page 557
Using the CRViewer object, Page 557

Adding the Report Viewer to a Visual Basic project

If you create a new report using the Create Report Expert in the Seagate 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 Seagate 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.
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 Seagate 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, Page 558*

*Working with secure data in reports, Page 558*

*Handling Report Viewer events, Page 559*

*Moving through a report, Page 559*

*Printing the report, Page 561*

*Controlling the appearance of the Report Viewer, Page 561*

*Connecting to the Web Reports Server, Page 562*

### 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.

```vbscript
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 Seagate 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 Seagate Crystal Web Reports Server.

Finally, the ViewReport method is called. This method has no parameters and has the job simply of displaying the specified report inside the Report Viewer control.

### Working with secure data in reports

If your report connects to a secure data source that requires log on information, you must release the Report object from the Report Viewer before you can log off of the data source. This can be done by assigning a new Report object to the ReportSource property, or by closing the CRViewer object. Until this is done, the data source will not be released from the Report object and you cannot log off.
Handling Report Viewer events

The Report Viewer control allows you to write custom code for several events relating to user interaction with both the control window and the report displayed. For instance, if you design a drill down report using the Report Designer Component, your users are likely to want to drill down on detail data. You can provide custom handling of such an event by writing code for the DrillOnGroup event.

To add event procedures to the Report Viewer control for the DrillOnGroup and PrintButtonClicked events:

1. In the Visual Basic Project window, select the Form containing the Report Viewer control.
2. Click the View Code button in the toolbar for the Project window. A code window for the form appears.
3. In the drop-down list box at the upper left hand corner of the code window, select the CRViewer1 control. (This name will appear different if you changed the Name property of the control in the Properties window.)
4. In the drop-down list box at the upper right corner of the code window, select the DrillOnGroup event. A procedure appears for handling the event.
5. Add the following code to the DrillOnGroup event procedure:

   ```vba
   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:

   ```vba
   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:

```vbnet
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:

```vbnet
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:

```vbnet
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.

```vbnet
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, Page 559.) 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.

*Note: The Enable properties cannot be assigned a value at runtime. They must be set using the Properties window at design time.*

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, Page 562,* and *Connecting from VBScript, Page 562,* 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.

```vbnet
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. For an example of this, refer to *Seagate Crystal Reports Web Administrator’s Guide.*
THE CRYSTAL REPORT VIEWER JAVA BEAN

The Seagate 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
What you will find in this chapter. . .

Report Viewer/ActiveX Object Model Technical Reference, Page 566
...including an outline of the Object Model hierarchy, and the Collections and Objects of CRVIEWERLib and CRWEBREPORTBROKERLib listed alphabetically with their properties, methods, and events.

The Report Viewer/Java Bean Technical Reference, Page 593
...including properties, methods, and events listed alphabetically.
The following diagram outlines the Report Viewer hierarchy.

```
CRViewer
  └── CRVEventInfo
  └── CRVTrackCursorInfo
```

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

- **CRField Object (CRVIEWERLib), Page 568**
  - CRField Object Properties, Page 568
- **CRFields Collection (CRVIEWERLib), Page 569**
  - CRFields Collection Properties, Page 569
- **CRVEventInfo Object (CRVIEWERLib), Page 569**
  - CRVEventInfo Object Properties, Page 569
  - CRVEventInfo Object Methods, Page 569
    - GetFields Method (CRVEventInfo Object), Page 569
- **CRViewer Object (CRVIEWERLib), Page 570**
  - CRViewer Object Properties, Page 570
  - CRViewer Object Methods, Page 571
    - ActivateView Method (CRViewer Object), Page 572
    - AddView Method (CRViewer Object), Page 572
    - CloseView Method (CRViewer Object), Page 572
    - GetCurrentPageNumber Method (CRViewer Object), Page 573
    - GetViewName Method (CRViewer Object), Page 573
    - GetViewPath Method (CRViewer Object), Page 573
    - PrintReport Method (CRViewer Object), Page 574
    - Refresh Method (CRViewer Object), Page 574
    - SearchByFormula Method (CRViewer Object), Page 574
— SearchForText Method (CRViewer Object), Page 575
— ShowFirstPage Method (CRViewer Object), Page 575
— ShowGroup Method (CRViewer Object), Page 575
— ShowLastPage Method (CRViewer Object), Page 576
— ShowNextPage Method (CRViewer Object), Page 576
— ShowNthPage Method (CRViewer Object), Page 576
— ShowPreviousPage Method (CRViewer Object), Page 576
— ViewReport Method (CRViewer Object), Page 576
— Zoom Method (CRViewer Object), Page 577

— CRViewer Object Events, Page 577
— Clicked Event (CRViewer Object), Page 578
— CloseButtonClicked Event (CRViewer Object), Page 578
— DblClicked Event (CRViewer Object), Page 579
— DownloadFinished Event (CRViewer Object), Page 579
— DownloadStarted Event (CRViewer Object), Page 579
— DrillOnDetail Event (CRViewer Object), Page 580
— DrillOnGraph Event (CRViewer Object), Page 580
— DrillOnGroup Event (CRViewer Object), Page 580
— DrillOnSubreport Event (CRViewer Object), Page 581
— ExportButtonClicked Event (CRViewer Object), Page 581
— FirstPageButtonClicked Event (CRViewer Object), Page 582
— GoToPageNClicked Event (CRViewer Object), Page 582
— GroupTreeButtonClicked Event (CRViewer Object), Page 582
— HelpButtonClicked Event (CRViewer Object), Page 583
— LastPageButtonClicked Event (CRViewer Object), Page 583
— NextPageButtonClicked Event (CRViewer Object), Page 583
— OnReportSourceError Event (CRViewer Object), Page 583
— PrevPageButtonClicked Event (CRViewer Object), Page 584
— PrintButtonClicked Event (CRViewer Object), Page 584
— RefreshButtonClicked Event (CRViewer Object), Page 584
CRField Object (CRVIEWERLib)

The CRField Object contains information related to the fields in a report displayed in the Report Viewer.

CRField Object Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>FieldType</td>
<td>CRFieldType (CRVIEWERLib) (page 590). Gets the type of field.</td>
<td>Read-only</td>
</tr>
<tr>
<td>IsRawData</td>
<td>Boolean. Gets whether or not the data in the field is raw data.</td>
<td>Read-only</td>
</tr>
<tr>
<td>Name</td>
<td>String. Gets the name of the field.</td>
<td>Read-only</td>
</tr>
<tr>
<td>Value</td>
<td>Variant. Default property gets the value in the field.</td>
<td>Read-only</td>
</tr>
</tbody>
</table>
CRFields Collection (CRVIEWERLib)

The CRFields Collection contains instances of CRFields Objects.

CRFields Collection Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Long. Gets the total number of items in the Collection.</td>
<td>Read-only</td>
</tr>
<tr>
<td>Item (index As Long)</td>
<td>Long. Default property gets the 1-based index number of the item in the Collection.</td>
<td>Read-only</td>
</tr>
<tr>
<td>SelectedFieldIndex</td>
<td>Long. Gets the index of the selected field.</td>
<td>Read-only</td>
</tr>
</tbody>
</table>

CRVEventInfo Object (CRVIEWERLib)

The CRVEventInfo Object contains information about events relating to objects within a report.

CRVEventInfo Object Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanDrillDown</td>
<td>Boolean. Gets whether or not the object is drillable.</td>
<td>Read-only</td>
</tr>
<tr>
<td>Index</td>
<td>Long. Gets or sets the number identifying a control in a control array.</td>
<td>Read-only</td>
</tr>
<tr>
<td>ParentIndex</td>
<td>Long. Gets reference to the object’s parent’s index.</td>
<td>Read-only</td>
</tr>
<tr>
<td>Text</td>
<td>String. Gets the object’s text string.</td>
<td>Read-only</td>
</tr>
<tr>
<td>Type</td>
<td>CRObjectType (CRViewerLib) (page 591). Gets the object type.</td>
<td>Read-only</td>
</tr>
</tbody>
</table>

CRVEventInfo Object Methods

The following methods are discussed in this section:

GetFields Method (CRVEventInfo Object), Page 569

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

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActiveViewIndex</td>
<td>Integer. Gets the 1-based index of the current view (tab).</td>
<td>Read only</td>
</tr>
<tr>
<td>DisplayBackgroundEdge</td>
<td>Boolean. Gets or sets whether the report is offset from the edge of its view window.</td>
<td>Read/Write</td>
</tr>
<tr>
<td>DisplayBorder</td>
<td>Boolean. Gets or sets whether the border of the viewer object is displayed.</td>
<td>Read/Write</td>
</tr>
<tr>
<td>DisplayGroupTree</td>
<td>Boolean. Gets or sets the visibility of the group tree.</td>
<td>Read/Write</td>
</tr>
<tr>
<td>DisplayTabs</td>
<td>Boolean. Gets or sets whether the viewer has tabs for navigation between views.</td>
<td>Read/Write</td>
</tr>
<tr>
<td>DisplayToolbar</td>
<td>Boolean. Gets or sets the visibility of the toolbar.</td>
<td>Read/Write</td>
</tr>
<tr>
<td>EnableAnimationCtrl</td>
<td>Boolean. Gets or sets whether or not the animation control is visible.</td>
<td>Read/Write</td>
</tr>
<tr>
<td>EnableCloseButton</td>
<td>Boolean. Gets or sets the visibility of the close button.</td>
<td>Read/Write</td>
</tr>
<tr>
<td>EnableDrillDown</td>
<td>Boolean. Gets or sets whether drill down is allowed.</td>
<td>Read/Write</td>
</tr>
<tr>
<td>EnableExportButton</td>
<td>Boolean. Gets or sets the visibility of the Export toolbar button.</td>
<td>Read/Write</td>
</tr>
<tr>
<td>EnableGroupTree</td>
<td>Boolean. Gets or sets whether or not the group tree is available.</td>
<td>Read/Write</td>
</tr>
<tr>
<td>EnableHelpButton</td>
<td>Boolean. Gets or sets whether or not the group tree is available.</td>
<td>Read/Write</td>
</tr>
<tr>
<td>EnableNavigationControls</td>
<td>Boolean. Gets or sets whether or not the help button appears on the toolbar.</td>
<td>Read/Write</td>
</tr>
<tr>
<td>EnablePopupMenu</td>
<td>Boolean. Gets or sets whether the popup menu is available. Design-time only.</td>
<td>Read/Write</td>
</tr>
<tr>
<td>EnablePrintButton</td>
<td>Boolean. Gets or sets the visibility of the Print button.</td>
<td>Read/Write</td>
</tr>
<tr>
<td>EnableProgressControl</td>
<td>Boolean. Gets or sets the visibility of the progress control.</td>
<td>Read/Write</td>
</tr>
<tr>
<td>EnableRefreshButton</td>
<td>Boolean. Gets or sets the visibility of the Refresh button.</td>
<td>Read/Write</td>
</tr>
</tbody>
</table>
Report Viewer Object Model

CRViewer Object Methods

The following methods are discussed in this section:

ActivateView Method (CRViewer Object), Page 572
AddView Method (CRViewer Object), Page 572
CloseView Method (CRViewer Object), Page 572
GetCurrentPageNumber Method (CRViewer Object), Page 573
GetViewName Method (CRViewer Object), Page 573
GetViewPath Method (CRViewer Object), Page 573
PrintReport Method (CRViewer Object), Page 574
Refresh Method (CRViewer Object), Page 574
SearchByFormula Method (CRViewer Object), Page 574
SearchForText Method (CRViewer Object), Page 575
ShowFirstPage Method (CRViewer Object), Page 575
ShowGroup Method (CRViewer Object), Page 575
ShowLastPage Method (CRViewer Object), Page 576
ShowNextPage Method (CRViewer Object), Page 576
ShowNthPage Method (CRViewer Object), Page 576

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnableSearchControl</td>
<td>Boolean. Gets or sets the visibility of the search control.</td>
<td>Read/Write</td>
</tr>
<tr>
<td>EnableSearchExpertButton</td>
<td>Boolean. Gets or sets the status of the Search Expert toolbar button.</td>
<td>Read/Write</td>
</tr>
<tr>
<td>EnableSelectExpertButton</td>
<td>Boolean. Gets or sets the status of the Select Expert toolbar button.</td>
<td>Read/Write</td>
</tr>
<tr>
<td>EnableStopButton</td>
<td>Boolean. Gets or sets whether the viewer displays the stop button.</td>
<td>Read/Write</td>
</tr>
<tr>
<td>EnableToolbar</td>
<td>Boolean. Gets or sets the visibility of the toolbar.</td>
<td>Read/Write</td>
</tr>
<tr>
<td>EnableZoomControl</td>
<td>Boolean. Gets or sets the visibility of the zoom control.</td>
<td>Read/Write</td>
</tr>
<tr>
<td>IsBusy</td>
<td>Boolean. Gets the status of the control, busy or not busy.</td>
<td>Read only</td>
</tr>
<tr>
<td>ReportSource</td>
<td>Unknown. Gets or sets the report source.</td>
<td>Read/Write</td>
</tr>
<tr>
<td>TrackCursorInfo</td>
<td>CRVTrackCursorInfo Object (CRVIEWERLib) (page 588). Gets reference to TrackCursor information.</td>
<td>Read only</td>
</tr>
<tr>
<td>ViewCount</td>
<td>Integer. Gets the current number of views (tabs).</td>
<td>Read only</td>
</tr>
</tbody>
</table>
ActivateView Method (CRViewer Object)

Use ActivateView method to activate a particular view.

Syntax

Sub ActivateView ( Index )

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The 1-based index number of the view that you want to activate.</td>
</tr>
</tbody>
</table>

AddView Method (CRViewer Object)

Use AddView method to add a new view tab to the Viewer.

Syntax

Sub AddView ( GroupPath )

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupPath</td>
<td>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.</td>
</tr>
</tbody>
</table>

CloseView Method (CRViewer Object)

Use CloseView method to close the specified view.

Syntax

Sub CloseView ( Index )

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The 1-based index number of the view that you want to close.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pTabName</td>
<td>Specifies the name of the current view (tab) As String.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>Specifies the 1-based index number of the view (tab) displayed in the Report Viewer for which you want to retrieve the path.</td>
</tr>
</tbody>
</table>
Report Viewers

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 )

574

Seagate Crystal Reports Technical Reference


### SearchForText Method (CRViewer Object)

Use `SearchForText` to search for the specified string.

**Syntax**

```vbs
Sub SearchForText ( Text As String )
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>Specifies the text that you want to search for As String.</td>
</tr>
</tbody>
</table>

### ShowFirstPage Method (CRViewer Object)

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

**Syntax**

```vbs
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**

```vbs
Sub ShowGroup (GroupPath)
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupPath</td>
<td>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).</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PageNumber</td>
<td>The page number that you want to display As Integer.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZoomLevel</td>
<td>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.</td>
</tr>
</tbody>
</table>

CRViewer Object Events
The following events are discussed in this section:

- Clicked Event (CRViewer Object), Page 578
- CloseButtonClick Event (CRViewer Object), Page 578
- DblClick Event (CRViewer Object), Page 579
- DownloadFinished Event (CRViewer Object), Page 579
- DownloadStarted Event (CRViewer Object), Page 579
- DrillOnDetail Event (CRViewer Object), Page 580
- DrillOnGraph Event (CRViewer Object), Page 580
- DrillOnGroup Event (CRViewer Object), Page 580
- DrillOnSubreport Event (CRViewer Object), Page 581
- ExportButtonClick Event (CRViewer Object), Page 581
- FirstPageButtonClick Event (CRViewer Object), Page 582
- GoToPageNClick Event (CRViewer Object), Page 582
- GroupTreeButtonClick Event (CRViewer Object), Page 582
- HelpButtonClick Event (CRViewer Object), Page 583
- LastButtonClick Event (CRViewer Object), Page 583
- NextButtonClick Event (CRViewer Object), Page 583
- OnReportSourceError Event (CRViewer Object), Page 583
- PrevButtonClick Event (CRViewer Object), Page 584
- PrintButtonClick Event (CRViewer Object), Page 584
RefreshButtonClicked Event (CRViewer Object), Page 584
SearchButtonClicked Event (CRViewer Object), Page 585
SearchExpertButtonClicked Event (CRViewer Object), Page 585
SelectionFormulaBuilt Event (CRViewer Object), Page 585
SelectionFormulaButtonClicked Event (CRViewer Object), Page 586
ShowGroup Event (CRViewer Object), Page 586
StopButtonClicked Event (CRViewer Object), Page 586
ViewChanged Event (CRViewer Object), Page 587
ViewChanging Event (CRViewer Object), Page 587
ZoomLevelChanged Event (CRViewer Object), Page 588

**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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Long. The X coordinate of the object clicked.</td>
</tr>
<tr>
<td>Y</td>
<td>Long. The Y coordinate of the object clicked.</td>
</tr>
<tr>
<td>EventInfo</td>
<td>A CRVEventInfo Object (CRVIEWERLib) (page 569), containing information about the object clicked.</td>
</tr>
<tr>
<td>UseDefault</td>
<td>Boolean. Indicates whether or not the default action of the event will be performed.</td>
</tr>
</tbody>
</table>

**CloseButtonClicked Event (CRViewer Object)**

The CloseButtonClicked event occurs when the Close Current View button is clicked.

**Syntax**

```
Event CloseButtonClicked ( UseDefault As Boolean )
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UseDefault</td>
<td>Boolean. Indicates whether or not the default action of the event will be performed.</td>
</tr>
</tbody>
</table>
**DblClicked Event (CRViewer Object)**

The DblClicked event occurs when an object is double clicked.

**Syntax**

```vbnet
Event DblClicked ( x As Long, y As Long, EventInfo, UseDefault As Boolean )
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Long. The X coordinate of the object that was double clicked.</td>
</tr>
<tr>
<td>Y</td>
<td>Long. The Y coordinate of the object that was double clicked.</td>
</tr>
<tr>
<td>EventInfo</td>
<td>A CRVEventInfo Object (CRVIEWERLib) (page 569), containing information about the object clicked.</td>
</tr>
<tr>
<td>UseDefault</td>
<td>Boolean. Indicates whether or not the default action of the event will be performed.</td>
</tr>
</tbody>
</table>

**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**

```vbnet
Event DownloadFinished ( loadingType As CRLoadingType )
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoadingType</td>
<td>CRLoadingType (CRViewerLib) (page 591). Indicates the type of data being loaded into the Report Viewer.</td>
</tr>
</tbody>
</table>

**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**

```vbnet
Event DownloadStarted ( loadingType As CRLoadingType )
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoadingType</td>
<td>CRLoadingType (CRViewerLib) (page 591). Indicates the type of data being loaded into the Report Viewer.</td>
</tr>
</tbody>
</table>
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
```vbnet
Event DrillOnDetail ( FieldValues, SelectedFieldIndex As Long, UseDefault As Boolean )
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FieldValues</td>
<td>An array of objects containing details on the field.</td>
</tr>
<tr>
<td>SelectedFieldIndex</td>
<td>Long. The array index of the value in the field actually drilled on.</td>
</tr>
<tr>
<td>UseDefault</td>
<td>Boolean. Indicates whether or not the default action of the event will be performed.</td>
</tr>
</tbody>
</table>

DrillOnGraph Event (CRViewer Object)
The DrillOnGraph event occurs when you drill down on a graph.

Syntax
```vbnet
Event DrillOnGraph ( PageNumber As Long, x As Long, y As Long, UseDefault As Boolean )
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PageNumber</td>
<td>Long. The page number of the report containing the graph where the event occurred.</td>
</tr>
<tr>
<td>x</td>
<td>Long. The X coordinate of the graph that was drilled on.</td>
</tr>
<tr>
<td>y</td>
<td>Long. The Y coordinate of the graph that was drilled on.</td>
</tr>
<tr>
<td>UseDefault</td>
<td>Boolean. Indicates whether or not the default action of the event will be performed.</td>
</tr>
</tbody>
</table>

DrillOnGroup Event (CRViewer Object)
The DrillOnGroup event occurs when drilling down (double-clicking) on a group field viewer window.

Syntax
```vbnet
Event DrillOnGroup ( GroupNameList, DrillType As CRDrillType, UseDefault As Boolean )
```
Report Viewers

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupNameList</td>
<td>An array containing all group names for the group drilled on.</td>
</tr>
<tr>
<td>DrillType</td>
<td>CRLoadingType (CRViewerLib) (page 591). Specifies what type of object the drill event occurred on (for example, graph, group tree, map, etc.).</td>
</tr>
<tr>
<td>UseDefault</td>
<td>Boolean. Indicates whether or not the default action of the event will be performed.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupNameList</td>
<td>An array containing all group names for the group drilled on.</td>
</tr>
<tr>
<td>SubreportName</td>
<td>String. Indicates the name of the subreport that was drilled on.</td>
</tr>
<tr>
<td>Title</td>
<td>String. Indicates the title of the subreport that was drilled on.</td>
</tr>
<tr>
<td>PageNumber</td>
<td>Long. Indicates the page number that the event occurred on.</td>
</tr>
<tr>
<td>Index</td>
<td>Long. Indicates the index of the subreport that was drilled on.</td>
</tr>
<tr>
<td>UseDefault</td>
<td>Boolean. Indicates whether or not the default action of the event will be performed.</td>
</tr>
</tbody>
</table>

ExportButtonClicked Event (CRViewer Object)
The ExportButtonClicked event occurs when the Export button is clicked.

Syntax

Event ExportButtonClicked (UseDefault As Boolean )

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UseDefault</td>
<td>Boolean. Indicates whether or not the default action of the event will be performed.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UseDefault</td>
<td>Boolean. Indicates whether or not the default action of the event will be performed.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UseDefault</td>
<td>Boolean. Indicates whether or not the default action of the event will be performed.</td>
</tr>
<tr>
<td>PageNumber</td>
<td>Integer. The page number of the page that is to be displayed.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visible</td>
<td>Boolean. Indicates whether or not the Group Tree is now visible.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UseDefault</td>
<td>Boolean. Indicates whether or not the default action of the event will be performed.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UseDefault</td>
<td>Boolean. Indicates whether or not the default action of the event will be performed.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>errorMsg</td>
<td>String. Indicates the error message.</td>
</tr>
<tr>
<td>errorCode</td>
<td>Long. Indicates the code or ID for the error.</td>
</tr>
<tr>
<td>UseDefault</td>
<td>Boolean. Indicates whether or not the default action of the event will be performed.</td>
</tr>
</tbody>
</table>

**PrevPageButtonClicked Event (CRViewer Object)**

The PrevPageButtonClicked event occurs when the button which navigates through the report to the previous page is clicked.

**Syntax**

```vb
Event PrevPageButtonClicked ( UseDefault As Boolean )
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UseDefault</td>
<td>Boolean. Indicates whether or not the default action of the event will be performed.</td>
</tr>
</tbody>
</table>

**PrintButtonClicked Event (CRViewer Object)**

The PrintButtonClicked event occurs when the Print button is clicked.

**Syntax**

```vb
Event PrintButtonClicked ( UseDefault As Boolean )
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UseDefault</td>
<td>Boolean. Indicates whether or not the default action of the event will be performed.</td>
</tr>
</tbody>
</table>

**RefreshButtonClicked Event (CRViewer Object)**

The RefreshButtonClicked event occurs when the Refresh button is clicked.

**Syntax**

```vb
Event RefreshButtonClicked ( UseDefault As Boolean )
```
**Report Viewer Object Model**

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UseDefault</td>
<td>Boolean. Indicates whether or not the default action of the event will be performed.</td>
</tr>
</tbody>
</table>

**SearchButtonClick Event (CRViewer Object)**

The SearchButtonClick event occurs when the Search button is clicked.

**Syntax**

```
Event SearchButtonClick ( searchText As String, UseDefault As Boolean )
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>searchText</td>
<td>String. Indicates the data being searched for.</td>
</tr>
<tr>
<td>UseDefault</td>
<td>Boolean. Indicates whether or not the default action of the event will be performed.</td>
</tr>
</tbody>
</table>

**SearchExpertButtonClick Event (CRViewer Object)**

The SearchExpertButtonClick event occurs when the Search Expert button is clicked.

**Syntax**

```
Event SearchExpertButtonClick ( UseDefault As Boolean )
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UseDefault</td>
<td>Boolean. Indicates whether or not the default action of the event will be performed.</td>
</tr>
</tbody>
</table>

**SelectionFormulaBuilt Event (CRViewer Object)**

The SelectionFormulaBuilt event occurs when a new selection formula is assigned to the report. This event is not valid when the Report Viewer is used in conjunction with the Report Designer Component.

**Syntax**

```
Event SelectionFormulaBuilt ( selectionFormula As String, UseDefault As Boolean )
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>selectionFormula</td>
<td>String. Indicates the new selection formula assigned to the report.</td>
</tr>
<tr>
<td>UseDefault</td>
<td>Boolean. Indicates whether or not the default action of the event will be performed.</td>
</tr>
</tbody>
</table>

SelectionFormulaButtonClick Event (CRViewer Object)

The SelectionFormulaButtonClick 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 SelectionFormulaButtonClick ( 
    selectionFormula As String, UseDefault As Boolean 
)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>selectionFormula</td>
<td>String. The current selection formula that will be replaced.</td>
</tr>
<tr>
<td>UseDefault</td>
<td>Boolean. Indicates whether or not the default action of the event will be performed.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupNameList</td>
<td>An array containing all group names for the group selected.</td>
</tr>
<tr>
<td>UseDefault</td>
<td>Boolean. Indicates whether or not the default action of the event will be performed.</td>
</tr>
</tbody>
</table>

StopButtonClick Event (CRViewer Object)

The StopButtonClick 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>loadingType</td>
<td>CRLoadingType (CRViewerLib) (page 591). Indicates what was being loaded into the Report Viewer when the Stop button was clicked.</td>
</tr>
<tr>
<td>UseDefault</td>
<td>Boolean. Indicates whether or not the default action of the event will be performed.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oldViewIndex</td>
<td>Long. An index referring to the view the user switched from.</td>
</tr>
<tr>
<td>newViewIndex</td>
<td>Long. An index referring to the view the user switched to.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oldViewIndex</td>
<td>Long. An index referring to the view the user is switching from.</td>
</tr>
<tr>
<td>newViewIndex</td>
<td>Long. An index referring to the view the user is switching to.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZoomLevel</td>
<td>Integer. A value indicating the new zoom level percentage.</td>
</tr>
</tbody>
</table>

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) (page 570).

CRVTrackCursorInfo Object Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>DetailAreaCursor</td>
<td>CRTrackCursor (CRViewerLib) (page 592). Gets or sets the DetailAreaCursor type.</td>
<td>Read/Write</td>
</tr>
<tr>
<td>DetailAreaFieldCursor</td>
<td>CRTrackCursor (CRViewerLib) (page 592). Gets or sets the DetailAreaFieldCursor type.</td>
<td>Read/Write</td>
</tr>
<tr>
<td>GraphCursor</td>
<td>CRTrackCursor (CRViewerLib) (page 592). Gets or sets the GraphCursor type.</td>
<td>Read/Write</td>
</tr>
<tr>
<td>GroupAreaCursor</td>
<td>CRTrackCursor (CRViewerLib) (page 592). Gets or sets the GroupAreaCursor type.</td>
<td>Read/Write</td>
</tr>
<tr>
<td>GroupAreaFieldCursor</td>
<td>CRTrackCursor (CRViewerLib) (page 592). Gets or sets the GroupAreaFieldCursor type.</td>
<td>Read/Write</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read/Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>ImageType</td>
<td>Reserved. Do not use in current development. Gets or sets the image type.</td>
<td>Read/Write</td>
</tr>
<tr>
<td>PromptOnRefresh</td>
<td>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.</td>
<td>Read/Write</td>
</tr>
<tr>
<td>ReportSource</td>
<td>Unknown. Gets or sets the report source for the WebReportBroker.</td>
<td>Read/Write</td>
</tr>
<tr>
<td>Title</td>
<td>String. Gets or sets the report title that can be displayed in viewer.</td>
<td>Read/Write</td>
</tr>
<tr>
<td>URL</td>
<td>String. Gets or sets the URL source for the report.</td>
<td>Read/Write</td>
</tr>
</tbody>
</table>

WebReportSource Object Methods

The following methods are discussed in this section:

AddParameter Method (WebReportSource Object), Page 589
AddParameterEx Method (WebReportSource Object), Page 590

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

```vbnet
Sub AddParameter ( tag As String, value As String )
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag</td>
<td>String. Specifies the prompt for which you want to pass a value.</td>
</tr>
<tr>
<td>value</td>
<td>String. Specifies the response string that you want to provide.</td>
</tr>
</tbody>
</table>

AddParameterEx Method (WebReportSource Object)

AddParameterEx method is not implemented for the current release. This method will allow you to pass more information than AddParameter Method, which should be used at this time.

Enumerated Types

The following enumerated types of the Report Viewer Object Model are discussed in this section:

CRLoadingType (CRViewerLib), Page 591
CRFieldType (CRViewerLib), Page 590
CRLoadingType (CRViewerLib), Page 591
CROBJECTType (CRViewerLib), Page 591
CRTrackCursor (CRViewerLib), Page 592

CRDrillType (CRViewerLib)

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crDrillOnGraph</td>
<td>2</td>
</tr>
<tr>
<td>crDrillOnGroup</td>
<td>0</td>
</tr>
<tr>
<td>crDrillOnGroupTree</td>
<td>1</td>
</tr>
<tr>
<td>crDrillOnMap</td>
<td>3</td>
</tr>
<tr>
<td>crDrillOnSubreport</td>
<td>4</td>
</tr>
</tbody>
</table>

CRFieldType (CRViewerLib)

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crBoolean</td>
<td>5</td>
</tr>
<tr>
<td>crCurrency</td>
<td>4</td>
</tr>
<tr>
<td>crDate</td>
<td>6</td>
</tr>
<tr>
<td>crDateTime</td>
<td>8</td>
</tr>
</tbody>
</table>
### CRLoadingType (CRViewerLib)

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoadingNothing</td>
<td>0</td>
</tr>
<tr>
<td>LoadingPages</td>
<td>1</td>
</tr>
<tr>
<td>LoadingQueryInfo</td>
<td>3</td>
</tr>
<tr>
<td>LoadingTotaller</td>
<td>2</td>
</tr>
</tbody>
</table>

### CRObjectType (CRViewerLib)

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crBitmap</td>
<td>103 (&amp;H67)</td>
</tr>
<tr>
<td>crBlob</td>
<td>104 (&amp;H68)</td>
</tr>
<tr>
<td>crBox</td>
<td>106 (&amp;H6A)</td>
</tr>
<tr>
<td>crCrossTab</td>
<td>110 (&amp;H6E)</td>
</tr>
<tr>
<td>crCrosstabChart</td>
<td>108 (&amp;H6C)</td>
</tr>
<tr>
<td>crCrosstabMap</td>
<td>115 (&amp;H73)</td>
</tr>
<tr>
<td>crDatabaseFieldType</td>
<td>1</td>
</tr>
<tr>
<td>crDetailChart</td>
<td>109 (&amp;H6D)</td>
</tr>
<tr>
<td>crDetailMap</td>
<td>116 (&amp;H74)</td>
</tr>
<tr>
<td>crDetailSection</td>
<td>202 (&amp;HCA)</td>
</tr>
<tr>
<td>crFormulaFieldType</td>
<td>5</td>
</tr>
<tr>
<td>crGraphic</td>
<td>111 (&amp;H6F)</td>
</tr>
<tr>
<td>crGroupChart</td>
<td>107 (&amp;H6B)</td>
</tr>
<tr>
<td>crGroupFooterSection</td>
<td>201 (&amp;HC9)</td>
</tr>
<tr>
<td>crGroupHeaderSection</td>
<td>200 (&amp;HC8)</td>
</tr>
<tr>
<td>crGroupMap</td>
<td>114 (&amp;HC8)</td>
</tr>
</tbody>
</table>
## Report Viewers

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crGroupNameFieldType</td>
<td>8</td>
</tr>
<tr>
<td>crLine</td>
<td>105 (&amp;H69)</td>
</tr>
<tr>
<td>crOLAPChart</td>
<td>113 (&amp;H71)</td>
</tr>
<tr>
<td>crOLAPCrossTabField</td>
<td>113 (&amp;H71)</td>
</tr>
<tr>
<td>crOLAPDataFieldType</td>
<td>3</td>
</tr>
<tr>
<td>crOLAPDimensionField</td>
<td>2</td>
</tr>
<tr>
<td>crOLAPMap</td>
<td>117 (&amp;H75)</td>
</tr>
<tr>
<td>crOLEObject</td>
<td>101 (&amp;H65)</td>
</tr>
<tr>
<td>crOOPSubreport</td>
<td>112 (&amp;H70)</td>
</tr>
<tr>
<td>crPageFooterSection</td>
<td>206 (&amp;HCE)</td>
</tr>
<tr>
<td>crPageHeaderSection</td>
<td>205 (&amp;HCD)</td>
</tr>
<tr>
<td>crPromptingVarField</td>
<td>9</td>
</tr>
<tr>
<td>crReportFooterSection</td>
<td>204 (&amp;HCC)</td>
</tr>
<tr>
<td>crReportHeaderSection</td>
<td>203 (&amp;HCB)</td>
</tr>
<tr>
<td>crSpecialVarFieldType</td>
<td>7</td>
</tr>
<tr>
<td>crSubreport</td>
<td>102 (&amp;H66)</td>
</tr>
<tr>
<td>crSummaryField</td>
<td>6</td>
</tr>
<tr>
<td>crTextEdit</td>
<td>100 (&amp;H64)</td>
</tr>
<tr>
<td>crUnknownFieldDefType</td>
<td>0</td>
</tr>
</tbody>
</table>

### CRTrackCursor (CRViewerLib)

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>crAppStartingCursor</td>
<td>12</td>
</tr>
<tr>
<td>crArrowCursor</td>
<td>1</td>
</tr>
<tr>
<td>crCrossCursor</td>
<td>2</td>
</tr>
<tr>
<td>crDefaultCursor</td>
<td>0</td>
</tr>
<tr>
<td>crHelpCursor</td>
<td>13</td>
</tr>
<tr>
<td>crIBeamCursor</td>
<td>3</td>
</tr>
<tr>
<td>crMagnifyCursor</td>
<td>99</td>
</tr>
<tr>
<td>crNoCursor</td>
<td>10</td>
</tr>
<tr>
<td>crWaitCursor</td>
<td>11</td>
</tr>
</tbody>
</table>
### 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.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Read(r), Write(w), Bound(b), Constrained(c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>busy</td>
<td>Boolean. True if the ReportViewer is currently processing a command initiated by user action, method call, or property change.</td>
<td>r, b</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Read(r), Write(w), Bound(b), Constrained(c)</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>canCloseCurrentView</td>
<td>Boolean. True if the current views tab can be closed. The initial (&quot;Preview&quot;) tab cannot be closed. Refer to method closeCurrentView.</td>
<td>r,b</td>
</tr>
<tr>
<td>canDrillDown</td>
<td>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.</td>
<td>r,w,b,c</td>
</tr>
<tr>
<td>currentMessage</td>
<td>string. The message currently displayed in the status area of the toolbar.</td>
<td>r,b</td>
</tr>
<tr>
<td>currentPageNumber</td>
<td>int. The number of the page currently being viewed.</td>
<td>r,b</td>
</tr>
<tr>
<td>currentTip</td>
<td>string. The &quot;tool tip&quot; currently displayed in the status area of the toolbar. Tool tips temporarily override any message in the status area.</td>
<td>r,b</td>
</tr>
<tr>
<td>currentViewName</td>
<td>string. The name of the view whose tab is selected.</td>
<td>r,b</td>
</tr>
<tr>
<td>exportingPossible</td>
<td>Boolean. False if exporting is not possible because the user has denied the bean permission to write to the local disk.</td>
<td>r</td>
</tr>
<tr>
<td>hasExportButton</td>
<td>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.</td>
<td>r,w,b,c</td>
</tr>
<tr>
<td>hasGroupTree</td>
<td>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).</td>
<td>r,w,b,c</td>
</tr>
<tr>
<td>hasPrintButton</td>
<td>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.</td>
<td>r,w,b,c</td>
</tr>
<tr>
<td>hasRefreshButton</td>
<td>Boolean. If True then the Refresh button is visible in the toolbar.</td>
<td>r,w,b,c</td>
</tr>
<tr>
<td>hasToolBar</td>
<td>Boolean. If True then the toolbar is visible.</td>
<td>r,w,b,c</td>
</tr>
<tr>
<td>hasTextSearchControls</td>
<td>Boolean. If True then the Text Search field and the Find Next button are visible in the toolbar.</td>
<td>r,w,b,c</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Read(r), Write(w), Bound(b), Constrained(c)</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------------------------------------</td>
</tr>
<tr>
<td>language</td>
<td>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</td>
<td>r,w,b,c</td>
</tr>
<tr>
<td>lastPageNumber</td>
<td>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).</td>
<td>r,b</td>
</tr>
<tr>
<td>lastPageNumberKnown</td>
<td>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.</td>
<td>r,b</td>
</tr>
<tr>
<td>printingPossible</td>
<td>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.</td>
<td>r</td>
</tr>
<tr>
<td>reportName</td>
<td>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.</td>
<td>r,w,b,c</td>
</tr>
<tr>
<td>searchText</td>
<td>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.</td>
<td>r,b</td>
</tr>
<tr>
<td>selectionFormula</td>
<td>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.</td>
<td>r,w,b,c</td>
</tr>
<tr>
<td>showGroupTree</td>
<td>Boolean. If True and the hasGroupTree property is True then the GroupTree will be visible.</td>
<td>r,w,b,c</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exportFormat</td>
<td>Specifies the format in which the requested report should appear.</td>
</tr>
<tr>
<td>Constant</td>
<td>Value</td>
</tr>
<tr>
<td>toHTML</td>
<td>0</td>
</tr>
<tr>
<td>toCrystalReport</td>
<td>1</td>
</tr>
<tr>
<td>toMSWord</td>
<td>2</td>
</tr>
<tr>
<td>toMSExcel</td>
<td>3</td>
</tr>
<tr>
<td>destinationFile</td>
<td>Complete pathname of the destination file.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>searchString</td>
<td>The string for which you want to search.</td>
</tr>
<tr>
<td>forwardSearch</td>
<td>This parameter is ignored.</td>
</tr>
<tr>
<td>caseSensitive</td>
<td>This parameter is ignored.</td>
</tr>
</tbody>
</table>

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

```java
void showPage ( int pageNumber )
```

**Parameter**

<table>
<thead>
<tr>
<th>pageNumber</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of the page to be displayed.</td>
<td></td>
</tr>
</tbody>
</table>

**stopAllCommands**

Cancels all unfinished commands. Equivalent to the Stop button in the toolbar.

Syntax

```java
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**

<table>
<thead>
<tr>
<th>Description</th>
<th>Properties</th>
<th>ListenerInterface Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indicates a request has been sent to the server owning the current report.</td>
<td>string ServerURL</td>
<td>requestStarted(event e)</td>
</tr>
<tr>
<td></td>
<td>string Parameter</td>
<td>requestEnde(event e)</td>
</tr>
</tbody>
</table>

**ViewChangeEvent**

<table>
<thead>
<tr>
<th>Description</th>
<th>Properties</th>
<th>ListenerInterface Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indicates that a view has changed.</td>
<td>string viewName</td>
<td>viewOpened(event e)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>viewActivated(event e)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>viewClosed(event e)</td>
</tr>
</tbody>
</table>
Crystal Data Source Object Models

What you will find in this chapter...

Crystal Data Objects, Page 600
...including a discussion of Crystal Data Objects and the Crystal ComObject with its Properties and Methods.
CrystalComObject Properties, Page 600
CrystalComObject Methods, Page 601

Crystal Data Source Type Library, Page 606
...including the CRDataSource COM Object and its Properties and Methods.
CRDataSource Properties, Page 607
CRDataSource Methods, Page 609
CRYSTAL DATA SOURCE OBJECT MODELS

Seagate 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, Page 65. 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 Active Data Driver, Page 56 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:

```vbnet
Dim cdoRowset As Object
Set cdoRowset = CreateObject("CrystalComObject.CrystalDataObject")
```

The following topics are discussed in this section.

- CrystalComObject Properties, Page 600
- CrystalComObject Methods, Page 601

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, Page 601
- AddRows, Page 602
- DeleteField, Page 602
- GetColCount, Page 603
- getEOF, Page 603
- GetFieldData, Page 603
- GetFieldName, Page 604
- GetFieldType, Page 604
- MoveFirst, Page 604
- MoveNext, Page 605
- MoveTo, Page 605
- Reset, Page 605

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(12, 4) 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, Page 601.

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, Page 601.

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 Active Data Driver, Page 56, 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, Page 69.

The following topics are discussed in this section.

CRDataSource Properties, Page 607

CRDataSource Methods, Page 609
**CRDataSource Properties**

The Crystal Data Source interface defines the following properties. An example of each property follows the property description.

- **Bookmark, Page 607**
- **EOF, Page 607**
- **FieldCount, Page 608**
- **FieldName, Page 608**
- **FieldType, Page 608**
- **FieldValue, Page 608**
- **RecordCount, Page 609**

**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**

```vba
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**

```vba
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, Page 609
MoveNext, Page 609

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, Page 607 to determine if the current record is the last record in the recordset.

Example

If Not myCRDataSource.CRDataSource_EOF Then
    myCRDataSource.CRDataSource_MoveNext
End If
Creating User-Defined Functions in C

What you will find in this chapter...

Overview of User-Defined Functions in C, Page 612
Programming User-Defined Functions in C, Page 612
  ...including an introduction and comments regarding function name, purpose, data types and return types.
Programming the UFL, Page 614
  ...including an introduction and topic outline as well as comments regarding example code; Helper Modules; Function Definition; Function Definition, Templates, Examples, and Error Tables; InitForJob and TermForJob Functions; UFL function implementation; returning user-defined errors; and obtaining parameter values from the parameter block.
Picture Function - a sample UFL function, Page 623
Module Definition (.def) File, Page 626
UFJOB Modules, Page 627
  ...including an introduction and
UFJOB.C, Page 627
UFJOB.H, Page 628
Implement InitJob and TermJob, Page 628
OVERVIEW OF USER-DEFINED FUNCTIONS IN C

The Seagate 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 Seagate 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, Page 629 or Creating User-Defined Functions in Delphi 3.0, Page 641.

PROGRAMMING USER-DEFINED FUNCTIONS IN C

You can add new functions to the Seagate 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 Seagate Crystal Reports Formula Editor, you need to determine the following:

- Name of the function, Page 612
- Purpose of the function, Page 613
- Function data, Page 613
  - the report data or user supplied data that the function will require.
- Return types, Page 614
  - 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:

(*** ) ***-****
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:

<table>
<thead>
<tr>
<th>Parameter Type (number)</th>
<th>C Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Double</td>
</tr>
<tr>
<td>currency</td>
<td>Double</td>
</tr>
<tr>
<td>Boolean</td>
<td>Short integer.</td>
</tr>
<tr>
<td>string</td>
<td>Pointer to an array of characters.</td>
</tr>
<tr>
<td>range (number)</td>
<td>Structure containing two doubles.</td>
</tr>
<tr>
<td>range (currency)</td>
<td>Structure containing two doubles.</td>
</tr>
<tr>
<td>range (Boolean)</td>
<td>Structure containing two short integers.</td>
</tr>
<tr>
<td>range (date)</td>
<td>Structure containing two long integers.</td>
</tr>
<tr>
<td>range (string)</td>
<td>Structure containing pointers to two elements in a character array.</td>
</tr>
<tr>
<td>array (number)</td>
<td>Pointer to a number array.</td>
</tr>
<tr>
<td>array (currency)</td>
<td>Pointer to a number array.</td>
</tr>
<tr>
<td>array (integer)</td>
<td>Pointer to an integer array.</td>
</tr>
<tr>
<td>array (Boolean)</td>
<td>Pointer to a Boolean array.</td>
</tr>
<tr>
<td>array (date)</td>
<td>Pointer to a date array.</td>
</tr>
<tr>
<td>array (string)</td>
<td>Pointer to a string array.</td>
</tr>
</tbody>
</table>
Return types

Finally, you must determine what kind of data your function returns to the current report in Seagate 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:

<table>
<thead>
<tr>
<th>Return Type</th>
<th>C Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>Pointer to a character array.</td>
</tr>
<tr>
<td>number</td>
<td>Double.</td>
</tr>
<tr>
<td>date</td>
<td>Long integer.</td>
</tr>
<tr>
<td>Boolean</td>
<td>Short integer.</td>
</tr>
<tr>
<td>currency</td>
<td>Double.</td>
</tr>
<tr>
<td>range (date)</td>
<td>Structure containing two long integers.</td>
</tr>
<tr>
<td>range (number)</td>
<td>Structure containing two doubles.</td>
</tr>
<tr>
<td>range (currency)</td>
<td>Structure containing two doubles.</td>
</tr>
<tr>
<td>range (string)</td>
<td>Structure containing pointers to two elements in a character array.</td>
</tr>
<tr>
<td>array (date)</td>
<td>Pointer to a date array.</td>
</tr>
<tr>
<td>array (number)</td>
<td>Pointer to a number array.</td>
</tr>
<tr>
<td>array (currency)</td>
<td>Pointer to a number array.</td>
</tr>
<tr>
<td>array (string)</td>
<td>Pointer to a string array.</td>
</tr>
<tr>
<td>array (Boolean)</td>
<td>Pointer to a Boolean array.</td>
</tr>
</tbody>
</table>

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 Seagate 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 Seagate Crystal Reports.

You can design your UFL from scratch, but you may find the Helper Modules, Page 615, provided with Seagate 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, Page 626, 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:

\[
\text{Picture}([\text{table.PHONENUM}], \ "(xxx) \ xxx-xxxx")
\]

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, Page 623, later in the chapter

The following topics are discussed in this section.

- Helper Modules, Page 615
- Setting Up a UFL Project, Page 616
- Function Definition, Page 617
- Function Definition Table, Page 618
  - Function definition table example, Page 618
- Function Templates Table, Page 619
- Function Examples Table, Page 619
- Error Table, Page 620
- InitForJob Function, Page 620
- TermForJob Function, Page 621
- UFL Function Implementation, Page 621
- Returning User-Defined Errors, Page 622
- Obtaining parameter values from the parameter block, Page 622

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, Page 627:

<table>
<thead>
<tr>
<th>File Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>ufddlh</td>
<td>Defines UFL enumerated, union, structure, and other data types.</td>
</tr>
<tr>
<td>ufuser.h</td>
<td>Prototypes of tables and functions user must implement.</td>
</tr>
</tbody>
</table>
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, Page 615, 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 Seagate Crystal Reports for Windows 3.x, or with a U2L prefix if it will be used with Seagate 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. Seagate 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 Seagate 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, Page 626. 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 UFLUNCS.PRJ, you would name your C module UFLSAMP.C or UFLUNCS.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 Seagate Crystal Reports when the Formula Editor accesses your UFL).

The private C code module of your UFL requires several parts:

- Function Definition Table, Page 618,
  — Function definition table example, Page 618
- Function Templates Table, Page 619,
- Function Examples Table, Page 619,
- Error Table, Page 620,
- InitForJob Function, Page 620,
- TermForJob Function, Page 621, and
- UFL Function Implementation, Page 621.

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 Seagate Crystal Reports Formula Editor.
Function Definition Table

You must supply a definition for each function that you want to add to Seagate 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.

Seagate 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 Seagate 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:

```c
UFFunctionExamples FunctionExamples [] =
{
    {"Picture (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:

```
"ErrorMessage"
```

*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, Page 622,* will examine how to handle parameter blocks.

All error strings must be set up in a table with the following heading:

```c
char *ErrorTable [] =
```

An error table for the Picture function should, at a minimum, look like this:

```c
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:

```c
void InitForJob(UFTInt32u jobID)
{
}
```

*Note: See the note following TermForJob Function, Page 621.*
**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:

```c
void TermForJob(UFTInt32u jobID)
{
}
```

*Note: Every UFL must have an implementation of InitForJob Function, Page 620, 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.*

Seagate 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, Page 628, which you can use to retrieve the job information for any open job. UFJOB Modules, Page 627, 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, Page 617), 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:

```c
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 Seagate 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 Seagate 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 Seagate 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 Seagate 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, Page 623), 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.
 *****/

Creating User-Defined Functions in C
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 [] =
{
    {"Picture (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)
{
}

l/*****
* 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-
ReturnValuelReturnString,
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 Seagate 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, Page 623):

```
Library     UFLSAMP
Description'User Function Library for Crystal Reports'
ExeType     Windows
HeapSize    1024
Code        Moveable Discardable Preload
Data        Moveable Preload
Segments    _TEXT Preload
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 Seagate Crystal Reports directory, \CRW, by default).

UFJOB MODULES

Two optional modules, UFJOB.C and UFJOB.H have been provided with Seagate Crystal Reports. These files provide an implementation of the InitForJob Function, Page 620, and TermForJob Function, Page 621, which allow you to obtain an ID number that is specific to the current print job in Seagate 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 Seagate Crystal Reports, demonstrate UFLs that group data according to the Top N values. (Seagate 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**, Page 627
  - add UFJOB.C to your UFL project file.
  - Note: See also UFJOB.H, Page 628
- **UFJOB.H**, Page 628
  - #include UFJOB.H in your private C code module that implements your UFL functions.
- Implement InitJob and TermJob, Page 628
  - 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, Page 620, and TermForJob Function, Page 621, that provide a JobInfo structure for the current job in Seagate 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:

```c
struct JobInfo *jobInfo;
jobInfo = FindJobInfo (ParamBlock->JobId);
```

UFJOB.H

This module prototypes the InitForJob Function, Page 620, TermForJob Function, Page 621, 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:

```c
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:

```c
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

What you will find in this chapter...

Overview of User-Defined Functions in Visual Basic, Page 630
Programming User-Defined Functions in Visual Basic, Page 630
    ...including an introduction and
    Using Visual Basic 4.0, Page 631
    Using Visual Basic 5.0, Page 633
    Registration of the Automation Server and Distribution of the Visual Basic Project, Page 634
    Using the User-Defined Functions, Page 634
Visual Basic and Seagate Crystal Reports, Page 635
    ...including comments regarding the following topics
    Variable Types, Page 636
    Using Arrays, Page 636
    Reserved Names, Page 636
    Function Name Prefixing, Page 637
    Passing Parameters By Reference and By Value, Page 638
    Error Handling, Page 638
    Special Purpose Functions, Page 638
Sample UFL Automation Server, Page 639
OVERVIEW OF USER-DEFINED FUNCTIONS IN VISUAL BASIC

The Seagate 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 Seagate 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, Page 612. For information on creating User-Defined Functions in an Automation Server using Delphi 3.0, see Programming User-Defined Functions in Delphi, Page 642.

PROGRAMMING USER-DEFINED FUNCTIONS IN VISUAL BASIC

The Seagate 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 Seagate Crystal Reports Professional Edition includes the User Function Library U2LCOM.DLL. This UFL is installed in your \WINDOWS\CRYSTAL directory when you install Seagate 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 Seagate 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.

The following topics are discussed in this section.

Using Visual Basic 4.0, Page 631
Using Visual Basic 5.0, Page 633
Registration of the Automation Server and Distribution of the Visual Basic Project, Page 634
Using the User-Defined Functions, Page 634

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:

1. Set Up the Main Subroutine, Page 631
2. Add a Class Module to the Project, Page 632
3. Add User Functions to the Class Module, Page 632
4. Name the Project, Page 632
5. Compile the Project as an OLE DLL, Page 633

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 Seagate 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 Seagate 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:

```vbnet
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 Seagate Crystal Reports Formula Editor interprets Visual Basic data types and arrays, see the sections *Visual Basic and Seagate Crystal Reports, Page 635* and *Using Arrays, Page 636*.

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 Project 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.

1. *Create a New ActiveX DLL Project, Page 633*
2. *Add User Functions to the Class Module, Page 633*
3. *Name the Project, Page 634*
4. *Build the DLL, Page 634*

**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 Seagate 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:

```vbnet
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 Seagate Crystal Reports Formula Editor interprets Visual Basic data types and arrays, see the sections *Visual Basic and Seagate Crystal Reports, Page 635*, and *Using Arrays, Page 636*.  

---

*Creating User-Defined Functions in Visual Basic*  

633
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 Seagate 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 Seagate Crystal Reports installed, then this file will already be located in the system’s \WINDOWS\CRYSTAL directory.

Using the User-Defined Functions

From Seagate 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 SEAGATE 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 Seagate 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).*

The following topics are discussed in this section.

*Variable Types*, Page 636
*Using Arrays*, Page 636
*Reserved Names*, Page 636
*Function Name Prefixing*, Page 637
*Passing Parameters By Reference and By Value*, Page 638
*Error Handling*, Page 638
*Special Purpose Functions*, Page 638
*Sample UFL Automation Server*, Page 639
Variable Types

Seagate 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 Seagate Crystal Reports converts the most common Visual Basic data types to data types supported by the Formula Editor:

<table>
<thead>
<tr>
<th>Visual Basic Data Types</th>
<th>Formula Editor Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>NumberVar</td>
</tr>
<tr>
<td>Long</td>
<td></td>
</tr>
<tr>
<td>Single</td>
<td></td>
</tr>
<tr>
<td>Double</td>
<td></td>
</tr>
<tr>
<td>Currency</td>
<td>CurrencyVar</td>
</tr>
<tr>
<td>Date</td>
<td>DateVar</td>
</tr>
<tr>
<td>Boolean</td>
<td>BooleanVar</td>
</tr>
<tr>
<td>String</td>
<td>StringVar</td>
</tr>
</tbody>
</table>

Note: Ranges--The range data type available in the Seagate 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 Seagate Crystal Reports Formula Editor. The following Visual Basic function is an acceptable User Defined Function for Seagate Crystal Reports:

```vbnet
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 Seagate Crystal Reports Formula Editor for special purposes:

- UFIinitialize
- UFTerminate
- UFStartJob
- UFEndJob
For more explanation of these function names, see Special Purpose Functions, Page 638. 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, Seagate 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:

<table>
<thead>
<tr>
<th>Project Name:</th>
<th>CRUFLTestFunctions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class Module Name:</td>
<td>Conversion</td>
</tr>
<tr>
<td>User Defined Function Name:</td>
<td>Date_To_String()</td>
</tr>
<tr>
<td>Formula Editor Function:</td>
<td>TestFunctionsConversionDateToString()</td>
</tr>
</tbody>
</table>

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:

```vba
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:

\[
\begin{align*}
\text{Public Function Test1 (num1 As Integer) As Integer} \\
\text{Public Function Test1 (ByRef num1 As Integer) As Integer} \\
\text{Public Function Test1 (ByVal num1 As Integer) As Integer}
\end{align*}
\]

Error Handling

If it is possible for your User Defined Function to produce an error, the Seagate 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 Seagate 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:

\[
\text{Public UFErrorText As String}
\]

Any time you trap for an error condition, simply set the UFErrorText property to the error text you want reported in Seagate Crystal Reports. Setting the value of this property triggers the error in Seagate Crystal Reports, and Seagate Crystal Reports displays a dialog box containing the error message that you assigned to UFErrorText.

\text{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 Seagate Crystal Reports Formula Editor can look for and process code defined in any of the following functions:

\[
\begin{align*}
\text{Public Function UFInitialize () As Long} \\
\text{Public Function UFTerminate () As Long} \\
\text{Public Sub UFStartJob (job As Long)} \\
\text{Public Sub UFEndJob (job As Long)}
\end{align*}
\]
Creating User-Defined Functions in Visual Basic

Be sure to define the functions in your code exactly as they appear above. If not defined correctly, they will be ignored by Seagate 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 Seagate Crystal Reports just before User-Defined Functions are evaluated (or reevaluated). The Seagate 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 Seagate 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 Seagate 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**

Seagate 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

What you will find in this chapter...

Overview of User-Defined Functions in Delphi, Page 642
Programming User-Defined Functions in Delphi, Page 642
...including an introduction and
Create the Project, Page 643
Create the Automation Object, Page 643
Add Methods to the Type Library, Page 643
Register the Type Library, Page 644
Create the User-Defined Functions, Page 644
Build the Project, Page 644
Using the User-Defined Functions, Page 644
Delphi and Seagate Crystal Reports, Page 645
...including comments regarding the following topics
Data Types, Page 646
Using Arrays, Page 646
Reserved Names, Page 646
Function Name Prefixing, Page 647
Passing Parameters By Reference and By Value, Page 647
Error Handling, Page 648
Special Purpose Functions, Page 648
Sample UFL, Page 649
OVERVIEW OF USER-DEFINED FUNCTIONS IN DELPHI

The Seagate 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 Seagate 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. 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, Page 612. For information on how to create User Defined Functions in an Automation Server using Visual Basic, see Programming User-Defined Functions in Visual Basic, Page 630.

PROGRAMMING USER-DEFINED FUNCTIONS IN DELPHI

The Seagate 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 Seagate Crystal Reports Professional Edition includes the User Function Library U2LCOM.DLL. This UFL is installed in your \WINDOWS\CRYSTAL directory when you install Seagate 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 Seagate 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.

*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:

1. **Create the Project, Page 643**
2. **Create the Automation Object, Page 643**
3. **Add Methods to the Type Library, Page 643**
4. **Register the Type Library, Page 644**
5. **Create the User-Defined Functions, Page 644**
6. **Build the Project, Page 644**
7. **Using the User-Defined Functions, Page 644**

**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 Seagate 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 Seagate Crystal Reports Formula Editor interprets Delphi data types and arrays, see Delphi and Seagate Crystal Reports, Page 645.

Continue creating methods for all User Defined Functions you want to appear in the Seagate 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 Seagate Crystal Reports Formula Editor.

Using the User-Defined Functions

From Seagate 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. Click OK and the Formula Editor appears.

4 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.

5 Double-click the User-Defined Function, and it appears in the Formula text box. Enter valid arguments for the function. For example:

   ProjectConversionSquare(5)

6 Click Check. Make sure no errors appear in the function.

7 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 SEAGATE 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 Seagate 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, Page 646
Using Arrays, Page 646
Reserved Names, Page 646
Function Name Prefixing, Page 647
Passing Parameters By Reference and By Value, Page 647
Error Handling, Page 648
Special Purpose Functions, Page 648
Data Types

Seagate 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 Seagate Crystal Reports converts the most common Delphi data types to data types supported by the Formula Editor:

<table>
<thead>
<tr>
<th>Delphi Data Types</th>
<th>Formula Editor Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>ShortInt</td>
<td>NumberVar</td>
</tr>
<tr>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>LongInt</td>
<td></td>
</tr>
<tr>
<td>Real</td>
<td></td>
</tr>
<tr>
<td>Single</td>
<td></td>
</tr>
<tr>
<td>Double</td>
<td></td>
</tr>
<tr>
<td>Currency</td>
<td>CurrencyVar</td>
</tr>
<tr>
<td>Date</td>
<td>DateVar</td>
</tr>
<tr>
<td>Boolean</td>
<td>BooleanVar</td>
</tr>
<tr>
<td>String</td>
<td>StringVar</td>
</tr>
</tbody>
</table>

Note: Ranges—The range data type available in the Seagate 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 Seagate Crystal Reports Formula Editor. The following Delphi function is an acceptable User-Defined Function for Seagate Crystal Reports:

```delphi
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 Seagate Crystal Reports Formula Editor for special purposes:

- UFInitialize
- UFTerminate
- UFStartJob
- UFEndJob
For more explanation of these function names, see Special Purpose Functions, Page 648.

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, Seagate 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:

<table>
<thead>
<tr>
<th>Project Name:</th>
<th>CRUFLTestFunctions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class Name:</td>
<td>Conversion</td>
</tr>
<tr>
<td>User Defined Function Name:</td>
<td>Date_To_String()</td>
</tr>
<tr>
<td>Formula Editor Function:</td>
<td>TestFunctionsConversionDateToString()</td>
</tr>
</tbody>
</table>

**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 Seagate 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 Seagate 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 Seagate Crystal Reports. Setting the value of this property triggers the error in Seagate Crystal Reports, and Seagate 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 Seagate 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 Seagate 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 Seagate Crystal Reports just before User-Defined Functions are evaluated (or reevaluated). The Seagate 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 Seagate 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 Seagate Crystal Reports job number as its only parameter. Handle any clean-up necessary on a per-job basis using this function.

**SAMPLE UFL**

A sample UFL project is located on the Seagate Crystal Reports installation CD. The self-extracting executable file UFLDELPHI.EXE is located in the \SAMPAPPS\COMUFLS\DELPHI directory on the CD. Simply run the executable file to install the sample project and all related files on your hard drive. This Delphi project creates three User-Defined Functions that can be used in the Seagate Crystal Reports Formula Editor.
What you will find in this appendix...

Product registration, Page 652
Product support, Page 652
Web support, Page 653
Email support, Page 654
Fax support, Page 654
Technical support, Page 654
Seagate Knowledge Link, Page 657
Product return policy, Page 657
Product replacement policy, Page 657
PRODUCT REGISTRATION

When using Seagate Crystal Reports for the first time, a dialog box appears that asks for your name, address, and related information. Then you are prompted to register the product to receive your Registration Number.

Registration methods:

- If you have access to the Internet, fill out the Product Registration form on the Seagate Software Web site at:
  
  fetch.seagatesoftware.com/register

  Or access the form from the Help menu by clicking Register/Change Address.

- Print the Product Registration form and then fax it to the Registration Fax number closest to you. Seagate will then fax back to you a registration number that can be entered into the product the next time you use it.

  Registration Fax Numbers

  USA/Canada +1 (604) 681-5147
  United Kingdom +44 (0) 181 231 0649
  Australia +6 2 9955 7682
  Hong Kong +852 2893 2727
  Singapore +65 777 8786

  Note: Fill out the Registration Form and mail it to Seagate Software. Seagate will mail a registration number back to you to enter into the product.

Registering the program will ensure that you are kept up-to-date with product advancements, and it will enable Seagate to provide quality technical support to people that are properly registered with Seagate Software.

When you enter the Registration Number, you have completed the registration process.

PRODUCT SUPPORT

Seagate Software is proud of the quality of its product and has spent a great deal of time trying to make it intuitive to use. If, however, you need assistance with something, consult the online Help system. The Help system can be accessed by clicking any Help button, by pressing the F1 key, or by selecting an option from the Help Menu. The Help system contains all of the information from the User’s Guide, as well as in-depth explanations and examples of all the reporting concepts.

  Note: Product support plans vary from region to region. Contact your Seagate Software Office/Distributor for a list of product support plans available in your region.

See Appendix B: Seagate Software International Office Directory, Page 659 for a list of Seagate Software International Offices.
WEB SUPPORT

Seagate Software Web site

To connect to the Seagate Software Web site:

1  Click Help.
2  Select Seagate Software on the Web.
3  Click Seagate Software Home Page.

Or enter the following address into your Web browser:
community.seagatesoftware.com

Note: “community” replaces “www” in the Web address.

Seagate Software Product Support

To connect to the Seagate Software Product Support Web page:

1  Click Help.
2  Select Seagate Software on the Web.
3  Click Online Support.

Or enter the following address into your Web browser:
community.seagatesoftware.com/support

Note: “community” replaces “www” in the Web address.

NEWSLETTER SERVICE

http://community.seagatesoftware.com/myprofile/subscribe.asp

Seagate Software’s Newsletter Service provides subscribers with information about product updates, current events, and new product releases. Each product you select will add your email address to that particular topic’s newsletter group. You may select multiple products and you may select both marketing and technical information. You will then see a confirmation page that simply asks you to verify your information. You may unsubscribe from any list at any time.
**EMAIL SUPPORT**

Email

answers@seagatesoftware.com

**To submit a Technical Support issue by email:**

Email a blank message to the address above. Leave everything blank and the system will send you back a form to complete.

**FAX SUPPORT**

Another efficient way to receive support for Seagate Crystal Reports is to fax your technical support request. To do this, first fill in the bottom part of the Technical Support Request form included with the product. This form appears when you choose the Technical Support Request from the Help Menu. When you have completed the form, click Print Request to print it.

Fax the completed form to Seagate Software at +1 (604) 681-7163. This service is available Monday through Friday (24 hours a day).

After a technical product specialist has had a chance to review your fax, they will respond to you by return fax the next business day.

**TECHNICAL SUPPORT**

Seagate Software offers several levels of telephone support for Seagate Crystal Reports.

**Priority Technical Support**

Seagate Crystal Reports Priority Support is designed to get you priority-queue response while saving you money on long-distance calling fees.

- Money-Saving Toll-Free Number**
- Priority-Call Response
- Extended Calling Hours**
- Online Services for 24-Hour Support
- Affordable One-Year Contract

**Money-Saving Toll-Free Number**

Get unlimited access to a toll-free number and save money on expensive long-distance calling times.
Priority-Call Response

We guarantee the fastest possible response times for Priority Support incidents. All Priority Support calls take precedence over Standard Support calls.

Extended Calling Hours**

Extended calling hours ensure you can get the answers you need, at the right time. We offer extended calling hours Monday to Friday from 8:00am - 8:00pm EST.

Online Services for 24-Hour Support

Seagate Software provides the tools for you to independently get fast answers to technical questions. We have a number of online services for unlimited email and Web based support, 24 hours a day, 7 days a week.

Affordable One-Year Contract

Sign-up for Seagate Crystal Reports Priority Support for a one-year contract from the date of purchase.

**Availability of toll-free numbers and extended calling hours varies by region.

Contacting Us

For more information, or to purchase Seagate Crystal Reports Priority Support, visit our Web site at www.seagatesoftware.com or contact us at the office nearest you.

Canada/USA: 1-800-877-2340 or 1-604-681-3435
Europe/Middle East/Africa: +44-181-566-2330
Asia/Pacific: +61-2-9955-4088

Standard Technical Support

Seagate Software provides customers with free phone, email and fax support for Seagate Crystal Reports users during regular business hours, Monday through Friday. Standard Support is free* to all registered users. We encourage you to register your product to ensure you have timely access to our technical support services.

*Free technical support is available for the first 60 days from the date of purchase.

<table>
<thead>
<tr>
<th>Location</th>
<th>Days</th>
<th>Times</th>
</tr>
</thead>
<tbody>
<tr>
<td>North America</td>
<td>M-F</td>
<td>11:00am - 7:00pm EST</td>
</tr>
<tr>
<td>+1 (604) 669-8379</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Austria</td>
<td>M-F</td>
<td>9:30am - 5:00pm CET</td>
</tr>
<tr>
<td>+ 43 (0) 1 79 5672 85</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Belgium</td>
<td>M-F</td>
<td>9:30am - 5:00pm CET</td>
</tr>
<tr>
<td>+ 32 (0) 2 713 12 35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>France</td>
<td>M-F</td>
<td>9:30am - 5:00pm CET</td>
</tr>
<tr>
<td>+33 (0) 141 918630</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Note: Long-distance charges may apply. Telephone numbers vary with location.

If you are calling from Africa or the Middle East, see Seagate Software International Office Directory, Page 659 to find out the telephone number of the support office closest to you.

If the country you are calling from is not listed above call the North American Standard Technical Support number at +1 (604) 669-8379.

Please have the following information available:

- **Product registration number.**
  
  If you are registered, you can find the product registration number by choosing About Crystal Report Designer from the Help Menu.
  
  If you are not registered, you will need to register to obtain the product registration number.

- **Product name and version number.**
  
  From the Help menu, click About Seagate Crystal Reports.

- **Operating system you are using.**
  
  For example, Windows 95, Windows 98, Windows NT, or Windows 3.11. Also whether you are using a 16-bit or 32-bit version of Seagate Crystal Reports.

- **Version of the database and other software you are using with Seagate Crystal Reports (if required).**
  
  Access, Btrieve, SQL, Sybase, Paradox, etc.

- **Network information, if you are on a network.**

- **Contents of AUTOEXEC.BAT and CONFIG.SYS files.**

- **The list of steps necessary to recreate the problem.**

- **The programming environment (if applicable).**

<table>
<thead>
<tr>
<th>Location</th>
<th>Days</th>
<th>Times</th>
</tr>
</thead>
<tbody>
<tr>
<td>Germany</td>
<td>M-F</td>
<td>9:30am - 5:00pm CET</td>
</tr>
<tr>
<td>+49 (0) 699 509 6180</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ireland</td>
<td>M-F</td>
<td>9:00am - 4:00pm GMT</td>
</tr>
<tr>
<td>+353 (0) 1 407 30 30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Netherlands</td>
<td>M-F</td>
<td>10:00am - 5:00pm CET</td>
</tr>
<tr>
<td>+31 (0) 20 346 9201</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sweden</td>
<td>M-F</td>
<td>10:00am - 5:00pm CET</td>
</tr>
<tr>
<td>+46 (0) 85 87 711 45</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Switzerland (German)</td>
<td>M-F</td>
<td>9:30am - 5:00pm CET</td>
</tr>
<tr>
<td>+41 (0) 1 800 9134</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Switzerland (French)</td>
<td>M-F</td>
<td>9:30am - 5:00pm CET</td>
</tr>
<tr>
<td>+41 (0) 1 800 9130</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UK</td>
<td>M-F</td>
<td>9:00am - 4:00pm GMT</td>
</tr>
<tr>
<td>+44 (0) 181 231 0638</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SEAGATE KNOWLEDGE LINK

Our Seagate Software Technical Product Specialists have created the Seagate Knowledge Link - our searchable database of technical reference articles. Shipped on CD, this application includes a base library of articles that can help new and experienced users resolve current technical issues. Seagate Knowledge Link includes 12 months of regular Technotes delivered bi-weekly via email.

Contacting Us

For more information, or to purchase Seagate Knowledge Link, visit our Web site at www.seagatesoftware.com or contact us at the office nearest you.

- Canada/USA: 1-800-877-2340 or 1-604-681-3435
- Europe/Middle East/Africa: +44-181-566-2330
- Asia/Pacific: +61-2-9955-4088

See the Seagate Software International Office Directory, Page 659 to find out the telephone number of the sales office closest to you.

PRODUCT RETURN POLICY

If you are not satisfied with Seagate Crystal Reports for any reason, you can return it to the original place of purchase for a refund within 30 days of the purchase date.

PRODUCT REPLACEMENT POLICY

If Seagate Crystal Reports diskettes or documentation are defective, then please contact Seagate Software, Inc., within 30 days of the purchase date. Fax a description of the problem and Seagate will solve it as quickly as possible. Please fax the description of the defect to (604) 681-2934.

Note: Replacement policies vary from region to region. Contact your local distributor for a list of replacement policies available in your region.
Seagate Software International Office Directory

What you will find in this appendix...

North/South American Head Office, Page 660
Asia/Pacific Offices, Page 661
Europe/Middle East/Africa Offices, Page 663
NORTH/SOUTH AMERICAN HEAD OFFICE

Canada & USA
Seagate Software, Inc.
840 Cambie Street
Vancouver, BC V6B 4J2
Canada

Sales
Telephone: 604-681-3435
Toll-Free: 1-800-877-2340
Fax: 604-681-2934
Email: answers@seagatesoftware.com

Product support
Phone: 604-669-8379
Fax: 604-681-7163
Email: answers@seagatesoftware.com (Available in English only)
Web browser: community.seagatesoftware.com/support (Available in English only)

Note: “community” replaces “www” in the Web address.

Mailing address: please refer to the address of the Seagate Software office nearest you.

Product registration
Fax: 604-681-5147
Email: answers@seagatesoftware.com
Web registration: fetch.seagatesoftware.com/register

Note: “fetch” replaces “www” in the Web address.

Mailing address: please see the addresses listed on the registration card included in your package.

Latin America
Seagate Software, Inc.
Suite 270 – 510 Thornall Street
Edison, NJ 08837
USA
Sales
Telephone: 732-321-6500
Fax: 732-321-6504

Product support
Please see the Product Support section listed under Canada.

Product registration
Please see the Product Registration section listed under Canada.

ASIA/PACIFIC OFFICES

Australia
Seagate Software Pty Ltd.
Level 4, 60 Albert Road
South Melbourne, VIC 3205
Australia

Sales
Telephone: +61 2 9955 4088
Sales +1-800-647-006
Fax: +61 2 9955 7682

Product support
Please see the Product Support section listed under Canada.

Product registration
Fax: +61 2 9955 7692
Mailing address: please see the addresses listed on the registration card included in your package.
Online registration: please see the product registration section listed under Canada.

Hong Kong
Seagate Software (HK)
Suite 2603, 26th Floor
Universal Trade Center
3 Arbuthnot Road, Central
Hong Kong
Sales
Telephone: +852 2575 2576
Fax: +852 2893 2727

Product support
Please see the Product Support section listed under Canada.

Product registration
Mailing address: please see the addresses listed on the registration card included in your package.
Online registration: please see the product registration section listed under Canada.

Japan
Seagate Software KK
Bridgestone Bldg. 2F
2-13-12 Hirakawa-cho,
Chiyoda-Ku,
Tokyo 102-0093

Sales
Telephone: +81.3. 5226. 3601
Fax: +81. 3 .5226 .3606
Email: sales_jp@img.seagatesoftware.com

Product support
Fax support: +81.3.5226.3605
Support email: support_jp@img.seagatesoftware.com

Product registration
Fax registration: +81.3.5226.3605
Email: answers@seagatesoftware.com
Web registration: fetch.seagatesoftware.com/register

Note: “fetch” replaces “www” in the Web address.
Mailing address: please see the addresses listed on the registration card included in your package.
Singapore
Seagate Software IMG
14 Science Park Drive
03-02 The Maxwell
Singapore Science Park
Singapore 118226

Sales
Telephone: +65 777 0533
Fax: +65 777 8786

Product support
Please see the Product Support section listed under Canada.

Product registration
Mailing address: please see the addresses listed on the registration card included in your package.
Online registration: please see the product registration section listed under Canada.

EUROPE/MIDDLE EAST/AFRICA OFFICES

United Kingdom - EMEA and Northern European Head Office
Seagate Software
The Broadwalk
54 The Broadway
Ealing, London
W5 5JN
UK

Sales
Telephone: +44 (0) 181 566 2330
Fax: +44 (0) 181 231 0600
Product support

<table>
<thead>
<tr>
<th>Country</th>
<th>Telephone</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>UK</td>
<td>+44 (0) 181 231 0638</td>
<td>English</td>
</tr>
<tr>
<td>Austria</td>
<td>+43 (0) 1 79 5672 85</td>
<td>German</td>
</tr>
<tr>
<td>Switzerland</td>
<td>+41 (0) 1 800 9134</td>
<td>German</td>
</tr>
<tr>
<td>Switzerland</td>
<td>+41 (0) 1 800 9130</td>
<td>French</td>
</tr>
<tr>
<td>Belgium</td>
<td>+32 (0) 2 713 12 35</td>
<td>French</td>
</tr>
<tr>
<td>Netherlands</td>
<td>+31 (0) 20 346 9201</td>
<td>English</td>
</tr>
<tr>
<td>Ireland</td>
<td>+353 (0) 1 407 30 30</td>
<td>English</td>
</tr>
<tr>
<td>Sweden</td>
<td>+46 (0) 85 87 711 45</td>
<td>English</td>
</tr>
</tbody>
</table>

Fax: +44 (0) 181 231 0600 (English, German, French)
Email: answers@seagatesoftware.com (available in English only).
Web browser: community.seagatesoftware.com/support (available in English only).

Note: “community” replaces “www” in the Web address.

Product registration

Fax: +44 (0) 181 231 0649
Email: answers@seagatesoftware.com
Web registration: fetch.seagatesoftware.com/register

Note: “fetch” replaces “www” in the Web address.

Mailing address: please see the addresses listed on the registration card included in your package.

Scotland

Seagate Software IMG
The Station Master’s Office
Dalmeny Station
South Queensferry West
Lothian
EH30 9JP
Scotland

Sales

Telephone: +44 (0) 1506 410444
Fax: +44 (0) 1506 414994
**Product support**
Please see the Product Support section listed under Northern European Head Office.

**Product registration**
Mailing address: please see the addresses listed on the registration card included in your package. 
Online registration: please see the product registration section listed under Northern Europe.

**Sweden**
Seagate Software
Kanalvagen 10C
194 61 Upplands Vasby
Sweden

**Sales**
Telephone: +46 8 590 04150
Fax: +46 8 590 04110

**Product support**
Mailing address: please see the addresses listed on the registration card included in your package.

**Product registration**
Please see the Product Registration section listed under Northern European Head Office.
Online registration: please see the product registration section listed under Northern Europe.

**Netherlands**
Seagate Software
Hojel City Center
Gebuw D, 4e etage
Graadt van Roggenweg 328
Postbox 19127
3501 DC Utrecht
The Netherlands

**Sales**
Telephone: +31 30 298 2124
Fax: +31 30 298 2125
**Product support**
Please see the Product Support section listed under Northern European Head Office.

**Product registration**
Mailing address: please see the addresses listed on the registration card included in your package.
Online registration: please see the product registration section listed under Northern Europe.

**Germany – Central European Head Office**
Seagate Software GmbH
2nd Floor
Frankfurter Strasse 21-25
D-65760 Eschborn
Germany

**Sales**
Telephone: +49 (0) 6196 9559 0
Fax: +49 (0) 6196 9559 10

**Product support**
Telephone: +49 (0) 699 509 6180
Email: answers@seagatesoftware.com (Available in English only)
Web browser: community.seagatesoftware.com/support (Available in English only)

**Product registration**
Fax: +49 (0) 6995 09 6182
Web browser: fetch.seagatesoftware.com/register

*Note: “fetch” replaces “www” in the Web address.*

Mailing address: please refer to the addresses listed on the registration card included in your package.

**Switzerland**
Seagate Software GmbH
World Trade Center
PO Box 112
Leutschenbachstrasse 95
CH-8050 Zurich
Switzerland
Sales
Telephone: +41 1 308 3922
Fax: +41 1 308 3500

Product support
Please see the Product Support section listed under Central European Head Office.

Product registration
Please see the addresses listed on the registration card included in your package.
Online registration: please see the product registration section listed under Central Europe.

France – Southern European Head Office
Seagate Software
62 bis, avenue Andre Morizet
F-92643 Boulogne Billancourt
Cedex
France

Sales
Telephone: +33 (0) 1 41 10 1600
Fax: +33 (0) 1 46 04 7419

Product support
Telephone: +33 (0) 141 918630
Fax +44 (0) 181 231 0600
Email: answers@seagatesoftware.com (Available in English only)
Web browser: community.seagatesoftware.com/support (Available in English only)

Product registration
Fax: ++ 33 (0) 1 41 91 86 27
Web browser: fetch.seagatesoftware.com/register
Mailing address: please refer to the addresses listed on the registration card included in your package.
Spain

Seagate Software IMG
Paseo de la Castellana 93 -4a
28046, Madrid
Spain

Sales
Telephone: +3491 555 5198
Fax: +3491 555 9957

Product support
Please see the Product Support section listed under Southern European Head Office.

Product registration
Mailing address: please see the addresses listed on the registration card included in your package.
Online registration: please see the product registration section listed under Southern Europe.

Italy

Seagate Software S.r.l.
Via Conservatorio 22
Milan 20122
Italy

Sales
Telephone: +39 02 7729 310
Fax: +39 02 7729 40

Product support
Please see the Product Support section listed under Southern European Head Office.

Product registration
Mailing address: please see the addresses listed on the registration card included in your package.
Online registration: please see the product registration section listed under Southern Europe.
South Africa Regional Office (Southern & Central Africa)

Seagate Software IMG
Lower Ground Floor
Block F, Pin Mill Farm
164 Katherine Street
Sandton
PO Box 786050
Sandton, 2146
Republic of South Africa

Sales
Telephone: +27 11 448 2080
Fax: +27 11 448 1960

Product support
Run locally in Johannesburg.
Telephone: +27 11 448 2080
Fax: +27 11 448 1960. Contact: Ross Harrison
Email: rossh@seagatesa.co.za
Email: answers@seagatesoftware.com (Available in English only)
Web browser: community.seagatesoftware.com/support (Available in English only)

Note: “community” replaces “www” in the Web address.

Product registration
Mailing address: please see the addresses listed on the registration card included in your package.
Online registration: please see the product registration section listed under Northern Europe.

Middle East Regional Office (ME & Northern Africa)

Seagate Software
PO Box 8687
Dubai
United Arab Emirates

Sales
Telephone: +971 4 523888
Fax: +971 4 519056
Product support
Run locally in Dubai.
Telephone: +971 4 523888 Contact : Katia Boueiri
Fax: +971 4 519056 Contact : Katia Boueiri
Email: kboueiri@img.seagatesoftware.com
Email: answers@seagatesoftware.com (Available in English only)
Web browser: community.seagatesoftware.com/support (Available in English only)

Note: “community” replaces “www” in the Web address.

Product registration
Mailing address: please see the addresses listed on the registration card included in your package.
Online registration: please see the product registration section listed under Northern Europe.
**INDEX**

<table>
<thead>
<tr>
<th>A</th>
<th>AddGraphObject method</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddField method</td>
<td>Report Designer Section Object</td>
</tr>
<tr>
<td>AddDefaultValue method</td>
<td>AddGroup method</td>
</tr>
<tr>
<td>AddCurrentRange method</td>
<td>AddLineObject method</td>
</tr>
<tr>
<td>AddCrossTabObject method</td>
<td>AddOLEDBSource method</td>
</tr>
<tr>
<td>AddBlobFieldObject method</td>
<td>AddParameter method</td>
</tr>
<tr>
<td>AddADOCommand method</td>
<td>AddParameterEx method</td>
</tr>
<tr>
<td>ActiveX Control</td>
<td>AddPieObject method</td>
</tr>
<tr>
<td>Crystal Data Object (CDO)</td>
<td>Report Viewer CRViewer Object</td>
</tr>
<tr>
<td>Crystal Smart Viewers</td>
<td>AddReportVariable method</td>
</tr>
<tr>
<td>Report Viewer Object Model</td>
<td>Report Designer Section Object</td>
</tr>
<tr>
<td>ActiveX Data Sources</td>
<td>Report Designer Section Object</td>
</tr>
<tr>
<td>using at design time</td>
<td>Report Designer Section Object</td>
</tr>
<tr>
<td>Add method</td>
<td>Report Designer Section Object</td>
</tr>
<tr>
<td>Report Designer CrossTabGroupsCollection</td>
<td>106</td>
</tr>
<tr>
<td>Report Designer DatabaseTablesCollection</td>
<td>121</td>
</tr>
<tr>
<td>Report Designer FieldDefinitionsCollection</td>
<td>127</td>
</tr>
<tr>
<td>Report Designer FormulaFieldDefinitions Collection</td>
<td>136</td>
</tr>
<tr>
<td>Report Designer ObjectSummaryFieldDefinitions Collection</td>
<td>149</td>
</tr>
<tr>
<td>Report Designer ParameterFieldDefinitions Collection</td>
<td>174</td>
</tr>
<tr>
<td>Report Designer RunningTotalFieldDefinitions Collection</td>
<td>194</td>
</tr>
<tr>
<td>Report Designer Sections Collection</td>
<td>207</td>
</tr>
<tr>
<td>Report Designer SortFields Collection</td>
<td>209</td>
</tr>
<tr>
<td>Report Designer SQLExpressionFieldDefinitions Collection</td>
<td>213</td>
</tr>
<tr>
<td>Report Designer SubreportLinksCollection</td>
<td>215</td>
</tr>
<tr>
<td>Report Designer SummaryFieldDefinitions Collection</td>
<td>221</td>
</tr>
<tr>
<td>Report Designer TableLinksCollection</td>
<td>223</td>
</tr>
<tr>
<td>AddTableLinks method</td>
<td>Report Designer WebReportSourceObject</td>
</tr>
<tr>
<td>AddBlobFieldObject method</td>
<td>ADO data sources</td>
</tr>
<tr>
<td>AddBoxObject method</td>
<td>AddReportVariable method</td>
</tr>
<tr>
<td>Report Designer Section Object</td>
<td>AddTextObject method</td>
</tr>
<tr>
<td>Report Designer Section Object</td>
<td>AddView method</td>
</tr>
<tr>
<td>Report Viewer CRViewer Object</td>
<td>AfterFormatPage Event</td>
</tr>
<tr>
<td>Address</td>
<td>Report Engine</td>
</tr>
<tr>
<td>Web site</td>
<td>Crystal Smart Viewers</td>
</tr>
<tr>
<td>application</td>
<td>Application Object</td>
</tr>
<tr>
<td>creating in VB</td>
<td>applications registering</td>
</tr>
<tr>
<td>error handling in VB</td>
<td>Report Engine</td>
</tr>
<tr>
<td>handling preview window events in VB</td>
<td>Automation Server</td>
</tr>
<tr>
<td>object name conflicts in VB</td>
<td>adding to VB project</td>
</tr>
<tr>
<td>obtaining in VB</td>
<td>Application Object</td>
</tr>
<tr>
<td>using in VB</td>
<td>distributing in VB applications</td>
</tr>
<tr>
<td></td>
<td>handling in VB</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Index-1
<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BeforeFormatPageEvent</td>
</tr>
<tr>
<td>Beans</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>sample application in VB</td>
</tr>
<tr>
<td>using in VB</td>
</tr>
<tr>
<td>AutoSetUnboundFieldSource method</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Collections
Areas, Report Designer ................................ 100
CRFields, Report Viewer ................................ 569
CrossTabGroups, Report Designer .................. 105
DatabaseFieldDefinitions, Report Designer ...... 116
DatabaseTables, Report Designer .................. 120
FieldDefinitions, Report Designer .................. 126
FormulaFieldDefinitions, Report Designer ....... 135
GroupNameFieldDefinitions, Report Designer ...... 144
ObjectSummaryFieldDefinitions, Report Designer . 149
Pages, Report Designer ................................ 166
ParameterFieldDefinitions, Report Designer ...... 173
Report Designer ......................................... 83
ReportObjects, Report Designer ..................... 189
RunningTotalFieldDefinitions, Report Designer ... 193
Sections, Report Designer ................................ 206
SoftFields, Report Designer ........................... 208
SQLExpressionFieldDefinitions, Report Designer ... 212
SubreportLinks, Report Designer .................... 214

Report Designer Report Object ...................... 182
Report Designer Application Object ................ 93
Report Designer FormulaFieldDefinition Object ... 135
Report Designer SQLExpressionFieldDefinition Object ................. 212
Report Designer ParameterFieldDefinition Object ................ 171
Report Viewer CRViewer Object ........................ 578
Report Viewer CRViewer Object ...................... 578
Report Viewer CRViewer Object ...................... 572
codes
section, see section codes

Areas, Report Designer ................................ 100
CRFields, Report Viewer ................................ 569
CrossTabGroups, Report Designer .................. 105
DatabaseFieldDefinitions, Report Designer ...... 116
DatabaseTables, Report Designer .................. 120
FieldDefinitions, Report Designer .................. 126
FormulaFieldDefinitions, Report Designer ....... 135
GroupNameFieldDefinitions, Report Designer ...... 144
ObjectSummaryFieldDefinitions, Report Designer . 149
Pages, Report Designer ................................ 166
ParameterFieldDefinitions, Report Designer ...... 173
Report Designer ......................................... 83
ReportObjects, Report Designer ..................... 189
RunningTotalFieldDefinitions, Report Designer ... 193
Sections, Report Designer ................................ 206
SoftFields, Report Designer ........................... 208
SQLExpressionFieldDefinitions, Report Designer ... 212
SubreportLinks, Report Designer .................... 214
SummaryFieldDefinitions, Report Designer ....... 220
TableLinks, Report Designer .......................... 223
COLORREF structure, Windows ..................... 523

constants
CRPE .................................................. 531
CRPE, Area/Section Format Formula .................. 533
CRPE, Chart Option, Bar Size .......................... 534
CRPE, Chart Option, Color ............................. 534
CRPE, Chart Option, Data Point ........................ 534
CRPE, Chart Option, Gridline ......................... 534
CRPE, Chart Option, Legend Layout .................... 534
CRPE, Chart Option, Legend Placement ............... 535
CRPE, Chart Option, Marker Shape .................... 535
CRPE, Chart Option, Marker Size ...................... 535
CRPE, Chart Option, Number Format ................... 535
CRPE, Chart Option, Pie Size ........................... 536
CRPE, Chart Option, Slice Detachment ................. 536
CRPE, Chart Option, Viewing Angle ..................... 536
CRPE, Chart Options .................................... 533
CRPE, Chart Subtype, 3D Riser ........................ 545
CRPE, Chart Subtype, 3D Surface ....................... 545
CRPE, Chart Subtype, Area ................................ 544
CRPE, Chart Subtype, Bar ................................ 543
CRPE, Chart Subtype, Bubble ............................ 545
CRPE, Chart Subtype, Doughnut ......................... 544
CRPE, Chart Subtype, Line ................................ 544
CRPE, Chart Subtype, Mstc ................................ 546
CRPE, Chart Subtype, Pie ................................ 544
CRPE, Chart Subtype, Radar ................................ 545
CRPE, Chart Subtype, Scatter ............................ 545
CRPE, Chart Subtype, Stock ................................ 546
CRPE, Database Type ..................................... 537
CRPE, Error Codes ....................................... 537
CRPE, Event Id ............................................ 541
CRPE, Field Mapping Type ................................ 542
CRPE, Formula Syntax .................................... 543
CRPE, Graph Subtype ..................................... 543
CRPE, Graph Text Font ................................... 546
CRPE, Graph Title Type ................................... 546
CRPE, Graph Type ........................................ 547
CRPE, Group Condition ................................... 547
CRPE, Group Condition, Boolean Type .................. 548
CRPE, Group Condition, Date and DateTimes ............... 548
CRPE, Group Condition, Mask and Type ................ 547
CRPE, Group Condition, Other Types ................... 548
CRPE, Job Destination .................................... 549
CRPE, Job Status .......................................... 549
CRPE, Object Type ....................................... 549
CRPE, Ole Object Type ................................... 550
CRPE, Ole Object Update .................................. 550
CRPE, Parameter Field Value Type ....................... 550
CRPE, Range Info ......................................... 551
CRPE, Section Codes ...................................... 551
CRPE, Sort Method ........................................ 551
CRPE, Sort Order ......................................... 551
CRPE, Track Cursor ....................................... 552
CRExportDestinationType Enum ................................. 531
Print Engine......................................................... 531
Controls
ActiveX .................................................................. 46
Grid .................................................................... 77
CRAlignment Enum
Report Designer .................................................... 229
CRMPMTEnum ......................................................... 229
CRAlignment Enum
Report Designer .................................................... 229
CRAreaKind Enum
Report Designer ..................................................... 230
CRBarSize Enum
Report Designer ..................................................... 230
CRBindingMatchTypeEnum
Report Designer ...................................................... 230
CRBooleanOutputType Enum
Report Designer ...................................................... 230
CRConvertDateTimeTypeEnum
Report Designer ..................................................... 231
CRCurrencyPositionTypeEnum
Report Designer ..................................................... 231
CRCurrencySymbolTypeEnum
Report Designer ..................................................... 231
CRDatabaseTypeEnum
Report Designer ..................................................... 231
CRDataSource ........................................................ 606
CRDateCalendarTypeEnum
Report Designer ..................................................... 231
CRDateEraTypeEnum
Report Designer ..................................................... 232
CRDateOrderEnum
Report Designer ..................................................... 232
CRDateWindowsDefaultTypeEnum
Report Designer ..................................................... 232
CRDayTypeEnum
Report Designer ..................................................... 232
CRDiscreteOrRangeKindEnum
Report Designer ..................................................... 232
CRDivisionMethodEnum
Report Designer ..................................................... 233
CRDrillTypeEnum
Report Viewer ....................................................... 590
Create Report Expert
database definition tool ........................................ 62
CreatePageGenerator method
Report Designer PageEngine Object ......................... 156
CreateSubreportPageGenerator method
Report Designer PageGenerator Object ..................... 160
creating
formatted bound reports ...................................... 79
CRExportDestinationTypeEnum
Report Designer ..................................................... 233
CRExportDestinationType Enum
Report Designer ..................................................... 233
CRExportFormatTypeEnum
Report Designer ..................................................... 234
CRExportDestinationType Enum
Report Designer ..................................................... 234
CRExportFormatTypeEnum
Report Designer ..................................................... 234
CRFieldKindEnum
Report Designer ..................................................... 235
CRFieldMappingTypeEnum
Report Designer ..................................................... 235
CRFieldTypeEnum
Report Viewer ....................................................... 590
CRFieldValueTypeEnum
Report Designer ..................................................... 235
CRFormulaSyntaxEnum
Report Designer ..................................................... 236
CRGraphColorEnum
Report Designer ..................................................... 236
CRGraphDataPointEnum
Report Designer ..................................................... 236
CRGraphDataTypeEnum
Report Designer ..................................................... 237
CRGraphJoinTypeEnum
Report Designer ..................................................... 237
CRGraphTypeEnum
Report Designer ..................................................... 237
CRGridColumnTypeEnum
Report Designer ..................................................... 238
CRGroupNameEnum
Report Designer ..................................................... 238
CRGroupConditionEnum
Report Designer ..................................................... 239
CRHourTypeEnum
Report Designer ..................................................... 239
CRImageTypeEnum
Report Designer ..................................................... 240
CRLeadingDayPositionEnum
Report Designer ..................................................... 240
CRLoadDataTypeEnum
Report Designer ..................................................... 240
CRLoadFromFileEnum
Report Designer ..................................................... 240
CRLoadPositionEnum
Report Designer ..................................................... 241
CRLineSpacingTypeEnum
Report Designer ..................................................... 241
CRLineStyleEnum
Report Designer ..................................................... 241
CRLinkJoinTypeEnum
Report Designer ..................................................... 241
CRLookupJoinTypeEnum
Report Designer ..................................................... 241
CRLookupJoinTypeEnum
Report Designer ..................................................... 242
CRLoadFromServerEnum
Report Viewer ....................................................... 591
CRMarkerShapeEnum
Report Designer ..................................................... 242
CRRadioButtonEnum
Report Designer ..................................................... 242
CRMargetTypeEnum
Report Designer ..................................................... 242
CRMonthEnum
Report Designer ..................................................... 243
Index-5
<table>
<thead>
<tr>
<th>CRNegativeType Enum</th>
<th>Report Designer</th>
<th>243</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRNumberFormat Enum</td>
<td>Report Designer</td>
<td>243</td>
</tr>
<tr>
<td>CRObjectKindEnum</td>
<td>Report Designer</td>
<td>244</td>
</tr>
<tr>
<td>CRObjectType Enum</td>
<td>Report Designer</td>
<td>591</td>
</tr>
<tr>
<td>CRPaperSize Enum</td>
<td>Report Designer</td>
<td>244</td>
</tr>
<tr>
<td>CRParameterFieldEnum</td>
<td>Report Designer</td>
<td>246</td>
</tr>
<tr>
<td>CRParameterFieldType</td>
<td>Report Designer</td>
<td>246</td>
</tr>
<tr>
<td>CRPaperSource Enum</td>
<td>Report Designer</td>
<td>246</td>
</tr>
<tr>
<td>CRPieEnum</td>
<td>Report Designer</td>
<td>246</td>
</tr>
<tr>
<td>CRPositiveType Enum</td>
<td>Report Designer</td>
<td>246</td>
</tr>
<tr>
<td>CRPieLegendLayoutEnum</td>
<td>Report Designer</td>
<td>247</td>
</tr>
<tr>
<td>CRPieSize Enum</td>
<td>Report Designer</td>
<td>247</td>
</tr>
<tr>
<td>CRPlaceHolderEnum</td>
<td>Report Designer</td>
<td>247</td>
</tr>
<tr>
<td>CRParameterPickListSortMethodEnum</td>
<td>Report Designer</td>
<td>247</td>
</tr>
<tr>
<td>CRParameterSortMethodEnum</td>
<td>Report Designer</td>
<td>247</td>
</tr>
<tr>
<td>CRPrintDuplexEnum</td>
<td>Report Designer</td>
<td>247</td>
</tr>
<tr>
<td>CRPrinterDuplexTypeEnum</td>
<td>Report Designer</td>
<td>247</td>
</tr>
<tr>
<td>CRPrintingProgressEnum</td>
<td>Report Designer</td>
<td>247</td>
</tr>
<tr>
<td>CRRangeInfo Enum</td>
<td>Report Designer</td>
<td>248</td>
</tr>
<tr>
<td>CRRenderResultEnum</td>
<td>Report Designer</td>
<td>248</td>
</tr>
<tr>
<td>CRRenderResultTypeEnum</td>
<td>Report Designer</td>
<td>248</td>
</tr>
<tr>
<td>CRReportFileFormatEnum</td>
<td>Report Designer</td>
<td>248</td>
</tr>
<tr>
<td>CRReportFileOrientationEnum</td>
<td>Report Designer</td>
<td>248</td>
</tr>
<tr>
<td>CRReportVariableValueEnum</td>
<td>Report Designer</td>
<td>248</td>
</tr>
<tr>
<td>CRReportVariableValueEnum</td>
<td>Report Designer</td>
<td>249</td>
</tr>
<tr>
<td>CRReportKindEnum</td>
<td>Report Designer</td>
<td>249</td>
</tr>
<tr>
<td>CRReportSummaryEnum</td>
<td>Report Designer</td>
<td>249</td>
</tr>
<tr>
<td>CRRoundingTypeEnum</td>
<td>Report Designer</td>
<td>249</td>
</tr>
<tr>
<td>CRRotationAngleEnum</td>
<td>Report Designer</td>
<td>249</td>
</tr>
<tr>
<td>CRRunningTotalConditionEnum</td>
<td>Report Designer</td>
<td>250</td>
</tr>
<tr>
<td>CRSearchDirectionEnum</td>
<td>Report Designer</td>
<td>250</td>
</tr>
<tr>
<td>CRSliceDetachmentEnum</td>
<td>Report Designer</td>
<td>250</td>
</tr>
<tr>
<td>CRSpecialVarTypeEnum</td>
<td>Report Designer</td>
<td>251</td>
</tr>
<tr>
<td>CRSpecialVarTypeEnum</td>
<td>Report Designer</td>
<td>251</td>
</tr>
<tr>
<td>CRTableDifferencesEnum</td>
<td>Report Designer</td>
<td>252</td>
</tr>
<tr>
<td>CRTextFormatEnum</td>
<td>Report Designer</td>
<td>253</td>
</tr>
<tr>
<td>CRTopOrBottomNGroupSortOrderEnum</td>
<td>Report Designer</td>
<td>253</td>
</tr>
<tr>
<td>CRTrackCursorEnum</td>
<td>Report Designer</td>
<td>254</td>
</tr>
<tr>
<td>CRViewingAngleEnum</td>
<td>Report Designer</td>
<td>254</td>
</tr>
<tr>
<td>CRYearTypeEnum</td>
<td>Report Designer</td>
<td>254</td>
</tr>
<tr>
<td>Crystal ActiveX Control</td>
<td>46</td>
<td></td>
</tr>
<tr>
<td>Crystal Data Object</td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>Crystal Data Object (CDO)</td>
<td>65</td>
<td></td>
</tr>
<tr>
<td>Crystal Data Object Model</td>
<td>68</td>
<td></td>
</tr>
<tr>
<td>Crystal Data Objects</td>
<td>600</td>
<td></td>
</tr>
<tr>
<td>Crystal Data Source Type Library</td>
<td>606</td>
<td></td>
</tr>
<tr>
<td>Crystal Data Source Type Library</td>
<td>69</td>
<td></td>
</tr>
<tr>
<td>Crystal Data Sources</td>
<td>76</td>
<td></td>
</tr>
<tr>
<td>Crystal Data Source object passing to Active Data Driver</td>
<td>75</td>
<td></td>
</tr>
<tr>
<td>Crystal Report Engine distributing applications</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>Crystal Report Engine API using</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Crystal Report Engine API using in Visual Basic</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Crystal Report Engine Automation Server see Automation Server</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>Crystal Report Engine Object Library viewing</td>
<td>53</td>
<td></td>
</tr>
<tr>
<td>Crystal Smart Viewer ActiveX adding Java Bean adding to VB project and secure project and secure data in reports appearance, controlling application development</td>
<td>556</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>PECheckSQLExpression</td>
<td>271</td>
<td></td>
</tr>
<tr>
<td>PEclearParameterCurrentValuesAndRanges</td>
<td>271</td>
<td></td>
</tr>
<tr>
<td>PEcloseParameterClickedEventInfo</td>
<td>443</td>
<td></td>
</tr>
<tr>
<td>PEcloseEngine</td>
<td>272</td>
<td></td>
</tr>
<tr>
<td>PEclosePrintJob</td>
<td>273</td>
<td></td>
</tr>
<tr>
<td>PEcloseSubreport</td>
<td>274</td>
<td></td>
</tr>
<tr>
<td>PEcloseWindow</td>
<td>274</td>
<td></td>
</tr>
<tr>
<td>PEConvertPFinInfoToVlnfo</td>
<td>275</td>
<td></td>
</tr>
<tr>
<td>PEConvertVlnfoToPFinfo</td>
<td>276</td>
<td></td>
</tr>
<tr>
<td>PEDeleteNthGroupSortField</td>
<td>277</td>
<td></td>
</tr>
<tr>
<td>PEDeleteNthParameterDefaultValue</td>
<td>278</td>
<td></td>
</tr>
<tr>
<td>PEDeleteNthSortField</td>
<td>279</td>
<td></td>
</tr>
<tr>
<td>PEDiscardSavedData</td>
<td>280</td>
<td></td>
</tr>
<tr>
<td>PEDrillOnDetailEventInfo</td>
<td>444</td>
<td></td>
</tr>
<tr>
<td>PEDrillOnGroupEventInfo</td>
<td>446</td>
<td></td>
</tr>
<tr>
<td>PEEnableEvent</td>
<td>281</td>
<td></td>
</tr>
<tr>
<td>PEEnableEventInfo structure</td>
<td>448</td>
<td></td>
</tr>
<tr>
<td>PEEnableProgressDialog</td>
<td>282</td>
<td></td>
</tr>
<tr>
<td>PEexportOptions structure</td>
<td>451</td>
<td></td>
</tr>
<tr>
<td>PEexportPrintWindow</td>
<td>283</td>
<td></td>
</tr>
<tr>
<td>PEexportTo</td>
<td>284</td>
<td></td>
</tr>
<tr>
<td>PEFieldsMappingEventInfo</td>
<td>452</td>
<td></td>
</tr>
<tr>
<td>PEFieldsValueInfo structure</td>
<td>453</td>
<td></td>
</tr>
<tr>
<td>PEFontColorInfo structure</td>
<td>455</td>
<td></td>
</tr>
<tr>
<td>PEGeneralPrintWindowEventInfo structure</td>
<td>458</td>
<td></td>
</tr>
<tr>
<td>PEGetAllowPromptDialog</td>
<td>285</td>
<td></td>
</tr>
<tr>
<td>PEGetAreaFormat</td>
<td>286</td>
<td></td>
</tr>
<tr>
<td>PEGetAreaFormatFormula</td>
<td>284</td>
<td></td>
</tr>
<tr>
<td>PEGetEnableEventInfo</td>
<td>287</td>
<td></td>
</tr>
<tr>
<td>PEGetAddressFormat</td>
<td>288</td>
<td></td>
</tr>
<tr>
<td>PEGetAddressErrorCode</td>
<td>289</td>
<td></td>
</tr>
<tr>
<td>PEGetAddressText</td>
<td>290</td>
<td></td>
</tr>
<tr>
<td>PEGetFieldMappingType</td>
<td>291</td>
<td></td>
</tr>
<tr>
<td>PEGetFormType</td>
<td>292</td>
<td></td>
</tr>
<tr>
<td>PEGetGraphsetFontInfo</td>
<td>293</td>
<td></td>
</tr>
<tr>
<td>PEGetGraphOptionInfo</td>
<td>294</td>
<td></td>
</tr>
<tr>
<td>PEGetGraphTextInfo</td>
<td>295</td>
<td></td>
</tr>
<tr>
<td>PEGetGraphTypeInfo</td>
<td>298</td>
<td></td>
</tr>
<tr>
<td>PEGetGroupPrivateInfo</td>
<td>299</td>
<td></td>
</tr>
<tr>
<td>PEGetGroupCondition</td>
<td>300</td>
<td></td>
</tr>
<tr>
<td>PEGetGroupOptions</td>
<td>301</td>
<td></td>
</tr>
<tr>
<td>PEGetGroupSelectionFormula</td>
<td>302</td>
<td></td>
</tr>
<tr>
<td>PEGetHandleString</td>
<td>304</td>
<td></td>
</tr>
<tr>
<td>PEGetJobStatus</td>
<td>305</td>
<td></td>
</tr>
<tr>
<td>PEGetMargins</td>
<td>306</td>
<td></td>
</tr>
<tr>
<td>PEGetNDetailCopies</td>
<td>306</td>
<td></td>
</tr>
<tr>
<td>PEGetNumberOfFormulas</td>
<td>307</td>
<td></td>
</tr>
<tr>
<td>PEGetNumberOfGroups</td>
<td>308</td>
<td></td>
</tr>
<tr>
<td>PEGetNumberOfSortFields</td>
<td>309</td>
<td></td>
</tr>
<tr>
<td>PEGetNumberOfPages</td>
<td>309</td>
<td></td>
</tr>
<tr>
<td>PEGetNumberOfParameterCurrentRanges</td>
<td>310</td>
<td></td>
</tr>
<tr>
<td>PEGetNumberOfParameterDefaultValues</td>
<td>311</td>
<td></td>
</tr>
<tr>
<td>PEGetNumberOfParameterFields</td>
<td>312</td>
<td></td>
</tr>
<tr>
<td>PEGetNumberOfParameters</td>
<td>313</td>
<td></td>
</tr>
<tr>
<td>PEGetNumberOfSections</td>
<td>314</td>
<td></td>
</tr>
<tr>
<td>PEGetNumberOfSortFields</td>
<td>315</td>
<td></td>
</tr>
<tr>
<td>PEGetPrintError</td>
<td>316</td>
<td></td>
</tr>
<tr>
<td>PEGetPrintOptions</td>
<td>317</td>
<td></td>
</tr>
<tr>
<td>PEGetPrintSubreportsInSection</td>
<td>317</td>
<td></td>
</tr>
<tr>
<td>PEGetPrintTables</td>
<td>318</td>
<td></td>
</tr>
<tr>
<td>PEGetPrintNthFormula</td>
<td>319</td>
<td></td>
</tr>
<tr>
<td>PEGetPrintNthGroupSortField</td>
<td>320</td>
<td></td>
</tr>
<tr>
<td>PEGetPrintNthParameterCurrentRange</td>
<td>321</td>
<td></td>
</tr>
<tr>
<td>PEGetPrintNthParameterCurrentValue</td>
<td>322</td>
<td></td>
</tr>
<tr>
<td>PEGetPrintNthParameterDefaultValue</td>
<td>324</td>
<td></td>
</tr>
<tr>
<td>PEGetPrintNthParameterField</td>
<td>325</td>
<td></td>
</tr>
<tr>
<td>PEGetPrintNthParameterType</td>
<td>326</td>
<td></td>
</tr>
<tr>
<td>PEGetPrintNthParameterValueDescription</td>
<td>327</td>
<td></td>
</tr>
<tr>
<td>PEGetPrintNthParameterValue</td>
<td>328</td>
<td></td>
</tr>
<tr>
<td>PEGetPrintNthSQLExpression</td>
<td>330</td>
<td></td>
</tr>
<tr>
<td>PEGetPrintSubreportInSection</td>
<td>330</td>
<td></td>
</tr>
<tr>
<td>PEGetPrintTableLocation</td>
<td>331</td>
<td></td>
</tr>
<tr>
<td>PEGetPrintTableLogOnInfo</td>
<td>332</td>
<td></td>
</tr>
<tr>
<td>PEGetPrintTablePrivateInfo</td>
<td>333</td>
<td></td>
</tr>
<tr>
<td>PEGetPrintTableSessionInfo</td>
<td>334</td>
<td></td>
</tr>
<tr>
<td>PEGetPrintTableType</td>
<td>335</td>
<td></td>
</tr>
<tr>
<td>PEGetPrintParameterMinMaxValue</td>
<td>337</td>
<td></td>
</tr>
<tr>
<td>PEGetPrintParameterPickListOption</td>
<td>338</td>
<td></td>
</tr>
<tr>
<td>PEGetPrintParameterValue</td>
<td>339</td>
<td></td>
</tr>
<tr>
<td>PEGetPrintDate</td>
<td>340</td>
<td></td>
</tr>
<tr>
<td>PEGetPrintOptions</td>
<td>341</td>
<td></td>
</tr>
<tr>
<td>PEGetReportOptions</td>
<td>341</td>
<td></td>
</tr>
<tr>
<td>PEGetReportSummaryInfo</td>
<td>342</td>
<td></td>
</tr>
<tr>
<td>PEGetReportTitle</td>
<td>343</td>
<td></td>
</tr>
<tr>
<td>PEGetSectionCode</td>
<td>344</td>
<td></td>
</tr>
<tr>
<td>PEGetSectionFormat</td>
<td>345</td>
<td></td>
</tr>
<tr>
<td>PEGetSectionFormatFormula</td>
<td>346</td>
<td></td>
</tr>
<tr>
<td>PEGetSectionHeight</td>
<td>347</td>
<td></td>
</tr>
<tr>
<td>PEGetSelectedPrinter</td>
<td>348</td>
<td></td>
</tr>
<tr>
<td>PEGetSelectionFormula</td>
<td>349</td>
<td></td>
</tr>
<tr>
<td>PEGetSQLExpression</td>
<td>350</td>
<td></td>
</tr>
<tr>
<td>PEGetSQLQuery</td>
<td>351</td>
<td></td>
</tr>
<tr>
<td>PEGetSQLServer</td>
<td>352</td>
<td></td>
</tr>
<tr>
<td>PEGetTrackCursorInfo</td>
<td>353</td>
<td></td>
</tr>
<tr>
<td>PEGetVersion</td>
<td>354</td>
<td></td>
</tr>
<tr>
<td>PEGetWindowHandle</td>
<td>355</td>
<td></td>
</tr>
<tr>
<td>PEGetWindowOptions</td>
<td>356</td>
<td></td>
</tr>
<tr>
<td>PEGraphAxisInfo structure</td>
<td>461</td>
<td></td>
</tr>
<tr>
<td>PEGraphAxisInfo</td>
<td>461</td>
<td></td>
</tr>
<tr>
<td>PEGraphOptionInfo structure</td>
<td>463</td>
<td></td>
</tr>
<tr>
<td>PEGraphOptionInfo</td>
<td>464</td>
<td></td>
</tr>
<tr>
<td>PEGraphTypeInfo</td>
<td>464</td>
<td></td>
</tr>
<tr>
<td>PEGroupOptions structure</td>
<td>467</td>
<td></td>
</tr>
<tr>
<td>PEGroupOptionsClickedEventInfo</td>
<td>468</td>
<td></td>
</tr>
<tr>
<td>PEGroupTreeButtonClickedEventInfo</td>
<td>468</td>
<td></td>
</tr>
<tr>
<td>PEHasSavedData</td>
<td>357</td>
<td></td>
</tr>
<tr>
<td>PEIsPrintJobFinished</td>
<td>358</td>
<td></td>
</tr>
<tr>
<td>PEJobInfoStruct</td>
<td>469</td>
<td></td>
</tr>
<tr>
<td>PELogOffServer</td>
<td>359</td>
<td></td>
</tr>
<tr>
<td>PELogOnInfoStruct</td>
<td>471</td>
<td></td>
</tr>
<tr>
<td>PELogOnServer</td>
<td>360</td>
<td></td>
</tr>
<tr>
<td>PELogOnServerWithPrivateInfo</td>
<td>362</td>
<td></td>
</tr>
<tr>
<td>PEMouseClickEventInfo</td>
<td>473</td>
<td></td>
</tr>
<tr>
<td>PEnextPrintWindowMagnification</td>
<td>473</td>
<td></td>
</tr>
<tr>
<td>PEOpenPrintWindowMagnification</td>
<td>473</td>
<td></td>
</tr>
<tr>
<td>PEOpenEngine</td>
<td>363</td>
<td></td>
</tr>
<tr>
<td>PEOpenPrintJob</td>
<td>364</td>
<td></td>
</tr>
</tbody>
</table>
Report Engine API ............................................................ 6
DEVMODE structure, Windows........................................... 523
discardSavedData method .................................................. 40
Report Designer Report Object ......................................... 183
distributing Report Engine applications ................................ 40
DLL
VB Wrapper .................................................................. 45
DownloadFinished Event .................................................. 579
DownloadStarted Event .................................................... 579
DrillOnDetail Event .......................................................... 580
Report Viewer CRViewer Object ....................................... 580
DrillOnGraph Event .......................................................... 580
Report Viewer CRViewer Object ....................................... 580
DrillOnGraph method ....................................................... 161
Report Designer PageGenerator Object ............................... 161
DrillOnGroup Event .......................................................... 580
Report Viewer CRViewer Object ....................................... 580
developers
what you should know .................................................. 3
development
Report Engine API ............................................................ 6
DEVMODE structure, Windows........................................... 523
discardSavedData method .................................................. 40
Report Designer Report Object ......................................... 183
distributing Report Engine applications ................................ 40
DLL
VB Wrapper .................................................................. 45
DownloadFinished Event .................................................. 579
DownloadStarted Event .................................................... 579
DrillOnDetail Event .......................................................... 580
Report Viewer CRViewer Object ....................................... 580
DrillOnGraph Event .......................................................... 580
Report Viewer CRViewer Object ....................................... 580
DrillOnGraph method ....................................................... 161
Report Designer PageGenerator Object ............................... 161
DrillOnGroup Event .......................................................... 580
Report Viewer CRViewer Object ....................................... 580

Enumerated Types

Drivers

DrillOnSubreport Event

Report Designer PageGenerator Object .................................. 162

DrillOnSubreport method

Report Viewer CRViewer Object ............................................. 581

DrillOnSubreport method

Report Designer PageGenerator Object .................................. 162

Drivers

Active Data ................................................................. 62
database (ADO, DAO, and RDO) ........................................... 56
using Active Data .......................................................... 57

email
technical support .................................................................. 654

Enumerated Types

CRAlignment, Report Designer ............................................... 229
CRAMFMType, Report Designer ............................................. 229
CRAreaKind, Report Designer ............................................... 230
CRBarSize, Report Designer .................................................. 230
CRBindingMatchType, Report Designer ................................... 230
CRBooleanOutputType, Report Designer ................................. 230
CRConvertDateTimeType, Report Designer ............................ 231
CRCurrencyPositionType, Report Designer ............................. 231
CRCurrencySymbolType, Report Designer .............................. 231
CRDateTimeType, Report Designer ......................................... 231
CRDateCalendarType, Report Designer ................................... 231
CRDateErType, Report Designer ............................................ 232
CRDateOrder, Report Designer ............................................. 232
CRDateWindowsDefaultType, Report Designer ....................... 232
CRDayType, Report Designer .............................................. 232
CRDiscreteOrRangeKind, Report Designer ............................... 233
CRDivisionMethod, Report Designer ...................................... 233
CRDateTimeType, Report Viewer ........................................... 590
CRExchangeDestinationType, Report Designer ......................... 233
CRExportDestinationType, Report Designer ........................... 234
CRExportFormatType, Report Designer .................................. 234
CRLFielKind, Report Designer ............................................. 235
CRLFielMappingType, Report Designer .................................... 235
CRLFielType, Report Viewer .................................................. 590
CRLFielValue, Report Designer ............................................ 235
CRFormulaSyntax, Report Designer ........................................ 236
CRGraphColor, Report Designer ............................................ 236
CRGraphDataPoint, Report Designer ...................................... 236
CRGraphDataType, Report Designer ....................................... 236
CRGraphDirection, Report Designer ...................................... 237
CRGraphType, Report Designer ............................................ 237
CRGroupName, Report Designer ............................................ 238
CRGroupCondition, Report Designer ...................................... 239
CRRHourType, Report Designer ............................................. 239
CRHTMLPageStyle, Report Designer ...................................... 240
CRHTMLToolbarStyle, Report Designer .................................. 240
CRImageType, Report Designer ............................................ 240
CRLDayType, Report Designer ............................................. 240
CRLegendPosition, Report Designer ...................................... 241

CRLineSpacingType, Report Designer ...................................... 241
CRLineStyle, Report Designer ............................................. 241
CRLinkJoinType, Report Designer ......................................... 241
CRLinkLookUpType, Report Designer ..................................... 242
CRLoadingType, Report Viewer ............................................ 591
CRMarkerShape, Report Designer ......................................... 242
CRMarkerSize, Report Designer .......................................... 242
CRMinuteType, Report Designer ........................................... 242
CRMonthType, Report Designer ............................................ 243
CRNegativeType, Report Designer ......................................... 243
CRNumberFormat, Report Designer ....................................... 243
CRObjectKind, Report Designer ........................................... 244
CRObjectType, Report Viewer ............................................. 591
CPRearrangement, Report Designer ....................................... 244
CPRSizing, Report Designer .............................................. 244
CRParameterField, Report Designer ...................................... 246
CRParameterPickListSortMethod, Report Designer .................. 246
CRPieLegendLayout, Report Designer ................................... 247
CRPieSize, Report Designer ............................................... 247
CRPlaceHolderType, Report Designer ..................................... 247
CRPrinterDuplexType, Report Designer .................................. 247
CRPrintingProgress, Report Designer .................................... 247
CRRangeInfo, Report Designer ............................................ 248
CRRangetype, Report Designer ............................................ 249
CRRoundType, Report Designer ............................................ 249
CRRunningTotalCondition, Report Designer ......................... 250
CRSearchDirection, Report Designer .................................... 250
CRSecondType, Report Designer ......................................... 250
CRSliceDetachment, Report Designer .................................... 250
CRSortDirection, Report Designer ........................................ 251
CRSpecialVarType, Report Designer ...................................... 251
CRSummaryType, Report Designer ........................................ 251
CRTableDiferences, Report Designer ..................................... 252
CRTextFormat, Report Designer ........................................... 253
CRTimeBase, Report Designer ............................................. 253
CRTopOrBottomNGroupSortOrder, Report Designer ................ 253

custom-print links .................................................................. 12

Error Table

See User-Defined Functions, errors in C.

User-Defined Functions in C .............................................. 622
User-Defined Functions in Delphi ....................................... 648
User-Defined Functions in VB ............................................ 638

establishing

Index-10
Events

- AfterFormatPage, Report Designer Report Object .......................... 188
- BeforeFormatPage, Report Designer Report Object .................. 188
- Clicked, Report Viewer CRViewer Object .............................. 578
- CloseButtonClicked, Report Viewer CRViewer Object .............. 578
- CRViewer Object, Report Viewer ......................................... 577
- DblClicked, Report Viewer CRViewer Object ......................... 579
- DownloadFinished, Report Viewer CRViewer Object .............. 579
- DownloadStarted, Report Viewer CRViewer Object ................. 579
- DrillOnDetail, Report Viewer CRViewer Object ...................... 580
- DrillOnGraph, Report Viewer CRViewer Object ...................... 580
- DrillOnGroup, Report Viewer CRViewer Object ...................... 580
- DrillOnSubreport, Report Viewer CRViewer Object ............... 581
- ExportButtonClicked, Report Viewer CRViewer Object .......... 581
- FieldMapping, Report Designer Report Object ...................... 581
- FirstPageButtonClicked, Report Viewer CRViewer Object ........ 582
- Format, Report Designer Section Object ............................... 205
- GoToPageNButtonClicked, Report Viewer CRViewer Object ....... 582
- GroupTreeButtonClicked, Report Viewer CRViewer Object ....... 582
- HelpButtonClicked, Report Viewer CRViewer Object .............. 582
- LastPageButtonClicked, Report Viewer CRViewer Object ......... 583
- NextPageButtonClicked, Report Viewer CRViewer Object ......... 583
- NoData, Report Designer Report Object ............................... 189
- OnReportSourceError, Report Viewer CRViewer Object ........ 583
- Preview window events in Automation Server ....................... 53
- PrevPageButtonClicked, Report Viewer CRViewer Object ........ 584
- PrintButtonClicked, Report Viewer CRViewer Object ............. 584
- RefreshButtonClicked, Report Viewer CRViewer Object .......... 584
- Report Object, Report Designer ......................................... 187
- Report Viewer, JavaBean .................................................. 598
- SearchButtonClicked, Report Viewer CRViewer Object .......... 584
- SearchExpertButtonClicked, Report Viewer CRViewer Object .... 585
- Section Object, Report Designer ........................................ 204
- SelectionFormulaBuilt, Report Viewer CRViewer Object .......... 585
- SelectionFormulaButtonClick, Report Viewer CRViewer Object ... 585
- ServerRequestEvent, Report Viewer JavaBean ...................... 598
- ShowGroup, Report Viewer CRViewer Object ....................... 586
- StopButtonClicked, Report Viewer CRViewer Object .............. 586
- ViewChanged, Report Viewer CRViewer Object .................... 587
- ViewChangeEvent, Report Viewer JavaBean ......................... 598
- ZoomLevelChanged, Report Viewer CRViewer Object ............. 587
- Crystal Smart Viewer ...................................................... 559
- events, preview window handling ...................................... 35
- export functions considerations for using ............................ 34
- Export method
  - Report Designer PageGenerator Object ............................ 163
  - Report Designer Report Object ...................................... 183
  - ExportButtonClicked .......................... 581
  - exporting reports ................................................. 32
  - CRPE ........................................... 256

F

- fax support .......................................................... 654

FieldMapping Event
- Report Designer Report Object ......................................... 188

Files

- Data Definition .......................................................... 57
- Data Definition, creating .............................................. 61
- .def, See Files, Module Definition, DLL, see DLL ................. 578
- Module Definition ..................................................... 626
- Module Definition, See also User-Defined Functions, programming in C ......................................................... 627
- UFJOB ................................................................. 34
- FindText method
  - Report Designer PageGenerator Object .......................... 163

Format Event
- Report Designer Section Object ...................................... 205
- formatted bound reports creating .................................... 79
- Function Definition Table, See User-Defined Functions, programming in C ......................................................... 79
- Function Definition, See User-Defined Functions, programming in C ......................................................... 79
- Function Example Table, See User-Defined Functions, programming in C ......................................................... 79
- Function Template Table, See User-Defined Functions, programming in C ......................................................... 79
- functions
  - CRPE ................................................................. 256
- Print Engine ........................................................... 256
Index-12

G
GetColCount method .......................... 603
GetCurrentPageNumber method
Report Viewer CRViewer Object ............. 573
GetEOF method
CrystalComObject.................................. 603
GetFieldData method
CrystalComObject.................................. 603
GetFieldName method
CrystalComObject.................................. 604
GetFields method
Report Viewer CRViewerInfo Object ........ 569
GetFieldType method
CrystalComObject.................................. 604
GetNextRows method
Report Designer Report Object ............. 183
GetNthCurrentRange method
Report Designer ParameterFieldDefinition Object ... 172
GetNthCurrentValue method
Report Designer ParameterFieldDefinition Object ... 172
GetNthDefaultValue method
Report Designer ParameterFieldDefinition Object ... 172
GetPageNumberForGroup method
Report Designer PageGenerator Object ...... 163
GetReportVariableValue method
Report Designer Report Object ............. 184
GetVersion method
Report Designer Application Object ....... 93
GetViewName method
Report Viewer CRViewer Object .......... 573
GetViewPath method
Report Viewer CRViewer Object .......... 573
GoToPageNButtonClick Event
Report Viewer CRViewer Object .......... 582

H
handling preview window events .............. 35
HelpButtonClicked Event
Report Viewer CRViewer Object .......... 583
Helper Modules .................................. 615
See also User-Defined Functions, programming in C.

I
implementing
InitForJob ........................................... 628
TermForJob ........................................... 628
ImportSubreport method
Report Designer Section Object .......... 204
InitForJob ........................................... 620
implementing ........................................ 628
See also Modules, UFJOB

See also User-Defined Functions, programming in C.

international office directory
Seagate Software ..................................... 659

Java
Crystal Smart Viewer applet .................. 564
Report Viewer Bean .............................. 565
Report Viewer Object Model ................ 593
Smart Viewer Bean ............................... 563

L
LastPageButtonClick Event
Report Viewer CRViewer Object .......... 583
Libraries
Crystal Data Source Type ....................... 69, 606
Crystal Report Engine Object .............. 53
UFL, See User-Defined Functions
links
coding custom-print .............................. 12
establishing custom-print ........................ 12
establishing print-only ............................ 9
LogOffServer method
Report Designer Application Object ....... 94
Report Designer Database Object .......... 110
LogOnServer method
Report Designer Application Object ....... 95
Report Designer Database Object .......... 111
LogOnServerEx method
Report Designer Application Object ....... 95
Report Designer Database Object .......... 112

M
methods
ActivateView, Report Viewer CRViewer Object ...... 572
Add, Report Designer CrossTabGroups Collection ... 106
Add, Report Designer DatabaseTables Collection ... 121
Add, Report Designer FieldDefinitions Collection ... 127
Add, Report Designer FormulaFieldDefinitions Collection ...................... 136
Add, Report Designer ObjectSummaryFieldDefinitions Collection .................. 149
Add, Report Designer ParameterFieldDefinitions Collection ...................... 174
Add, Report Designer RunningTotalFieldDefinitions Collection ................... 194
Add, Report Designer Sections Collection .......... 207
Add, Report Designer SortFields Collection .......... 209
Add, Report Designer SQLExpressionFieldDefinitions Collection .................. 213
Add, Report Designer SubreportLinks Collection ...... 215
Add, Report Designer SummaryFieldDefinitions Collection ...................... 221
Add, Report Designer TableLinks Collection .......... 223
MoveFirst method
CRDataSource ........................................ 609
CrystalComObject .................................... 604
MoveNext method
CRDataSource ........................................ 609
CrystalComObject .................................... 605
MoveTo method
CrystalComObject .................................... 605

N
NewReport method
Report Designer Application Object ............... 96
NextPageButtonClicked Event
Report Viewer CRViewer Object ..................... 583
NoData Event
Report Designer Report Object ...................... 189

O
Object Model
ActiveX Report Viewer .................................. 565
Crystal Data Objects .................................... 600
Crystal Data Source ..................................... 600
CrystalComObject ....................................... 600
Report Designer ........................................ 82

Objects
Application, Report Designer ......................... 92
Area, Report Designer ................................ 98
Automation Server Application, see Application Object
Automation Server Report, see Report Object
BlobFieldObject, Report Designer ..................... 101
BoxObject, Report Designer .......................... 103
CRField, Report Viewer ................................ 568
CrossTabGroup, Report Designer ...................... 104
CrossTabObject, Report Designer ...................... 106
CRVEventInfo, Report Viewer .......................... 569
CRViewer ................................................... 557

Index-15

example code for .......................................................... 11
PrintOut method
Report Designer Report Object ........................................ 185
PrintReport method
Report Viewer CRViewer Object ...................................... 574
Product
registration ................................................................. 652
replacement policy ........................................................ 657
return policy ................................................................. 657
Programming
Report Engine API .......................................................... 6
User-Defined Functions in C ............................................ 612, 614
User-Defined Functions in Delphi .................................... 642
User-Defined Functions in VB ......................................... 630
PromptForExportOptions method
Report Designer ExportOptions Object ................................ 126
Properties
Area Object, Report Designer .............................................. 98
Areas Collections, Report Designer ....................................... 101
BlobFieldObject Object, Report Designer ......................... 101
BoxObject Object, Report Designer .................................... 103
CRField Object, Report Viewer ........................................... 563
CRFields Collection, Report Viewer .................................. 569
CrossTabGroup Object, Report Designer ............................ 104
CrossTabGroups Collection, Report Designer .................... 105
CrossTabObject, Report Designer ....................................... 106
CRVEventInfo Object, Report Viewer .................................. 569
CRVViewer Object, Report Viewer ...................................... 570
Database Object, Report Designer ..................................... 109
DatabaseFieldDefinition Object, Report Designer ............. 115
DatabaseFieldDefinitions Collection, Report Designer .... 116
Designer ........................................................................... 116
DatabaseTable Object, Report Designer ............................ 116
DatabaseTables Collection, Report Designer ................... 121
ExportOptions Object, Report Designer ............................. 123
FieldDefinitions Collection, Report Designer ................. 126
FieldMappingData Object, Report Designer ..................... 127, 128
FormattingInfo Object, Report Designer ......................... 133
FormulaFieldDefinition Object, Report Designer ............ 134
FormulaFieldDefinitions Collection, Report Designer .... 136
Designer ........................................................................... 136
GraphObject Object, Report Designer ............................... 137
GroupNameFieldDefinition Object, Report Designer ........ 143
GroupNameFieldDefinitions Collection, Report Designer .... 144
IFieldDefinition Object, Report Designer ............................ 144
IReportObject Object, Report Designer .............................. 145
LineObject Object, Report Designer ................................. 146
MapObject Object, Report Designer ................................. 147
ObjectSummaryFieldDefinitions Collection, Report Designer .. 149
OlapGridObject Object, Report Designer ........................... 150
OLEObject Object, Report Designer .................................... 151
Page Object, Report Designer ............................................. 154
| GraphObject Object                      | 137 |
| GraphObject Object Properties          | 137 |
| GroupNameFieldDefinition Object        | 143 |
| GroupNameFieldDefinition Object Properties | 143 |
| GroupNameFieldDefinitions Collection   | 144 |
| GroupNameFieldDefinitions Collection Properties | 144 |
| IFieldDefinition Object               | 144 |
| IFieldDefinition Object Properties     | 144 |
| IReportObject Object                  | 145 |
| IReportObject Object Properties        | 145 |
| LineObject Object                     | 146 |
| LineObject Object Properties           | 146 |
| MapObject Object                      | 147 |
| MapObject Object Properties            | 147 |
| Object                                      | 82 |
| object naming conflicts                  | 83 |
| Objects                                    | 83 |
| ObjectSummaryFieldDefinitions Collection | 149 |
| ObjectSummaryFieldDefinitions Collection methods | 149 |
| ParameterFieldDefinition Object         | 150 |
| ParameterFieldDefinition Object Properties | 150 |
| OLEObject Object                       | 151 |
| OLEObject Object methods                | 153 |
| OLEObject Object Properties             | 151 |
| Page Object                             | 154 |
| Page Object methods                     | 154 |
| Page Object Properties                  | 154 |
| PageEngine Object                      | 156 |
| PageEngine Object methods               | 156 |
| PageEngine Object Properties            | 156 |
| PageGenerator Object                    | 159 |
| PageGenerator Object methods            | 160 |
| PageGenerator Object Properties         | 159 |
| Pages Collection                        | 166 |
| Pages Collection Properties             | 166 |
| ParameterFieldDefinition Object         | 166 |
| ParameterFieldDefinition Object methods  | 170 |
| ParameterFieldDefinition Object Properties | 167 |
| ParameterFieldDefinitions Collection    | 173 |
| ParameterFieldDefinitions Collection methods | 174 |
| ParameterFieldDefinitions Collection Properties | 173 |
| PrintingStatus Object                   | 175 |
| PrintingStatus Object Properties        | 175 |
| Report Object                           | 175 |
| Report Object Events                    | 187 |
| Report Object methods                   | 180 |
| Report Object Properties                | 175 |
| ReportObjects Collection                | 189 |
| ReportObjects Collection Properties     | 189 |
| RunningTotalFieldDefinition Object      | 190 |
| RunningTotalFieldDefinition Object methods | 191 |
| RunningTotalFieldDefinition Object Properties | 190 |
| RunningTotalFieldDefinitions Collection | 193 |
| RunningTotalFieldDefinitions Collection methods | 194 |
| RunningTotalFieldDefinitions Collection Properties | 194 |
| Section Object                          | 195 |
| Section Object Events                   | 204 |
| Section Object methods                  | 196 |
| Section Object Properties               | 195 |
| Sections Collection                     | 206 |
| Sections Collection methods             | 207 |
| Sections Collection Properties          | 206 |
| SortField Object                        | 208 |
| SortField Object Properties             | 208 |
| SortFields Collection                   | 208 |
| SortFields Collection Methods           | 209 |
| SpecialVarFieldDefinition Object        | 210 |
| SpecialVarFieldDefinition Object Properties | 210 |
| SQLExpressionFieldDefinition Object     | 211 |
| SQLExpressionFieldDefinition Object methods | 212 |
| SQLExpressionFieldDefinition Object Properties | 211 |
| SQLExpressionFieldDefinitions Collection | 212 |
| SQLExpressionFieldDefinitions Collection methods | 213 |
| SQLExpressionFieldDefinitions Collection Properties | 212 |
| SubreportLink Object                    | 214 |
| SubreportLink Object Properties         | 214 |
| SubreportLinks Collection               | 214 |
| SubreportLinks Collection methods       | 215 |
| SubreportLinks Collection Properties    | 214 |
| SubreportObject Object                  | 216 |
| SubreportObject Object methods          | 217 |
| SubreportObject Object Properties       | 216 |
| SummaryFieldDefinition Object           | 218 |
| SummaryFieldDefinition Object Properties | 218 |
| SummaryFieldDefinitions Collection      | 221 |
| SummaryFieldDefinitions Collection Methods | 220 |
| TableLink Object                        | 222 |
| TableLink Object Properties             | 222 |
| TableLinks Collection                   | 223 |
| TableLinks Collection Methods           | 223 |
| TextObject Object                       | 224 |
| TextObject Object methods               | 226 |
| TextObject Object Properties            | 224 |

**Report Engine**

- **API** ................................................. 6
- before using in your application .................. 3
- distributing applications ......................... 40
- introduction ........................................... 2
- using ...................................................... 4

**Report Engine API**

- overview ............................................. 6
- structures .............................................. 30
- using in Visual Basic ................................. 42
- variable length strings ............................. 27
<table>
<thead>
<tr>
<th>Subject</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>errors in VB</td>
<td>638</td>
</tr>
<tr>
<td>function name prefixing in Delphi</td>
<td>647</td>
</tr>
<tr>
<td>function name prefixing in VB</td>
<td>637</td>
</tr>
<tr>
<td>implementing in C</td>
<td>621</td>
</tr>
<tr>
<td>naming in C</td>
<td>612</td>
</tr>
<tr>
<td>overview in C</td>
<td>612</td>
</tr>
<tr>
<td>overview in Delphi</td>
<td>642</td>
</tr>
<tr>
<td>overview in VB</td>
<td>630</td>
</tr>
<tr>
<td>passing parameters by reference and by value in VB</td>
<td>638</td>
</tr>
<tr>
<td>passing parameters by value and by reference in Delphi</td>
<td>647</td>
</tr>
<tr>
<td>programming in C</td>
<td>612–628</td>
</tr>
<tr>
<td>programming in Delphi</td>
<td>642</td>
</tr>
<tr>
<td>reserved names in Delphi</td>
<td>646</td>
</tr>
<tr>
<td>reserved names in VB</td>
<td>636</td>
</tr>
<tr>
<td>sample Automation Server in VB</td>
<td>639</td>
</tr>
<tr>
<td>sample in C</td>
<td>649</td>
</tr>
<tr>
<td>special purpose functions in Delphi</td>
<td>648</td>
</tr>
<tr>
<td>special purpose functions in VB</td>
<td>638</td>
</tr>
<tr>
<td>User-Defined Functions in C</td>
<td>611</td>
</tr>
<tr>
<td>User-Defined Functions in VB</td>
<td>641</td>
</tr>
<tr>
<td>using in VB</td>
<td>634</td>
</tr>
<tr>
<td>variable types in VB</td>
<td>636</td>
</tr>
<tr>
<td>using</td>
<td>42</td>
</tr>
<tr>
<td>Crystal Report Engine API in Visual Basic</td>
<td>4</td>
</tr>
<tr>
<td>the Crystal Report Engine API</td>
<td>8</td>
</tr>
<tr>
<td>UXDDiskOptions structure</td>
<td>512</td>
</tr>
<tr>
<td>UXDMAPIOptions structure</td>
<td>513</td>
</tr>
<tr>
<td>UXPostFolderOptions structure</td>
<td>515</td>
</tr>
<tr>
<td>UXDSMIOptions structure</td>
<td>514</td>
</tr>
<tr>
<td>UXDVIMOptions structure</td>
<td>516</td>
</tr>
<tr>
<td>UXCharSeparatedOptions structure</td>
<td>517</td>
</tr>
<tr>
<td>UXCommaTabSeparatedOptions structure</td>
<td>518</td>
</tr>
<tr>
<td>UXDelimitedOptions structure</td>
<td>519</td>
</tr>
<tr>
<td>UXHTML3Options structure</td>
<td>520</td>
</tr>
<tr>
<td>UXODBCOptions structure</td>
<td>520</td>
</tr>
<tr>
<td>UXPaginatedTextOptions structure</td>
<td>521</td>
</tr>
<tr>
<td>UXRecordStyleOptions structure</td>
<td>522</td>
</tr>
<tr>
<td>V</td>
<td></td>
</tr>
<tr>
<td>variable length strings</td>
<td>27</td>
</tr>
<tr>
<td>VB syntax</td>
<td></td>
</tr>
<tr>
<td>PEAddParameterCurrentRange</td>
<td>262</td>
</tr>
<tr>
<td>PEAddParameterCurrentValue</td>
<td>264</td>
</tr>
<tr>
<td>PEAddParameterDefaultValue</td>
<td>265</td>
</tr>
<tr>
<td>PEAddParameterCurrentValuesAndRanges</td>
<td>271</td>
</tr>
<tr>
<td>PEAddParameterDefaultValuesAndRanges</td>
<td>271</td>
</tr>
<tr>
<td>PEAddParameterCurrentRangeAndRanges</td>
<td>271</td>
</tr>
<tr>
<td>PEAddParameterDefaultValueAndRanges</td>
<td>271</td>
</tr>
<tr>
<td>PECancelPrintJob</td>
<td>265</td>
</tr>
<tr>
<td>PECancelCloseEngine</td>
<td>266</td>
</tr>
<tr>
<td>PECancelGroupSelectionFormula</td>
<td>268</td>
</tr>
<tr>
<td>PECancelNthTableDifferences</td>
<td>269</td>
</tr>
<tr>
<td>PEClearParameterCurrentValue</td>
<td>264</td>
</tr>
<tr>
<td>PEClearParameterCurrentValuesAndRanges</td>
<td>271</td>
</tr>
<tr>
<td>PECloseButtonClickedEventInfo</td>
<td>443</td>
</tr>
<tr>
<td>PECloseEngine</td>
<td>272</td>
</tr>
<tr>
<td>PEClosePrintJob</td>
<td>273</td>
</tr>
<tr>
<td>PECloseSubreport</td>
<td>274</td>
</tr>
<tr>
<td>PECloseWindow</td>
<td>274</td>
</tr>
<tr>
<td>PECoverPageOptions</td>
<td>271</td>
</tr>
<tr>
<td>PEDeleteNthGroupSortField</td>
<td>277</td>
</tr>
<tr>
<td>PEDeleteNthParameterDefaultValue</td>
<td>278</td>
</tr>
<tr>
<td>PEDeleteNthSortField</td>
<td>279</td>
</tr>
<tr>
<td>PEDiscardSavedData</td>
<td>280</td>
</tr>
<tr>
<td>PEDrillOnGroupEventInfo</td>
<td>446</td>
</tr>
<tr>
<td>PEEnableEventInfo</td>
<td>447</td>
</tr>
<tr>
<td>PEEnableProgressDialog</td>
<td>282</td>
</tr>
<tr>
<td>PEExportOptions structure</td>
<td>451</td>
</tr>
<tr>
<td>PEExportWindow</td>
<td>283</td>
</tr>
<tr>
<td>PEExportTo</td>
<td>283</td>
</tr>
<tr>
<td>PEFieldValueInfo</td>
<td>453</td>
</tr>
<tr>
<td>PEFontColorInfo</td>
<td>455</td>
</tr>
<tr>
<td>PEFormulaSyntax structure</td>
<td>456</td>
</tr>
<tr>
<td>PEGetHandleString</td>
<td>284</td>
</tr>
<tr>
<td>PEGeneralPrintWindowEventInfo structure</td>
<td>457</td>
</tr>
<tr>
<td>PEGetAllowPromptDialog</td>
<td>285</td>
</tr>
<tr>
<td>PEGetAreaFormat</td>
<td>286</td>
</tr>
<tr>
<td>PEGetAreaFormatFormula</td>
<td>287</td>
</tr>
<tr>
<td>PEGetCharCode</td>
<td>288</td>
</tr>
<tr>
<td>PEGetErrorText</td>
<td>289</td>
</tr>
<tr>
<td>PEGetExportOptions</td>
<td>290</td>
</tr>
<tr>
<td>PEGetFieldMappingType</td>
<td>291</td>
</tr>
<tr>
<td>PEGetFormula</td>
<td>292</td>
</tr>
<tr>
<td>PEGetFormulaSyntax</td>
<td>293</td>
</tr>
<tr>
<td>PEGetGraphAxisInfo</td>
<td>295</td>
</tr>
<tr>
<td>PEGetGraphFontInfo</td>
<td>294</td>
</tr>
<tr>
<td>PEGetGraphOptionInfo</td>
<td>296</td>
</tr>
<tr>
<td>PEGetGraphTextDefaultOption</td>
<td>297</td>
</tr>
<tr>
<td>PEGetGraphTextInfo</td>
<td>298</td>
</tr>
<tr>
<td>PEGetGroupCondition</td>
<td>300</td>
</tr>
<tr>
<td>PEGetGroupOptions</td>
<td>301</td>
</tr>
<tr>
<td>PEGetGroupSelectionFormula</td>
<td>302</td>
</tr>
<tr>
<td>PEGetHandleString</td>
<td>303</td>
</tr>
<tr>
<td>PEGetJobStatus</td>
<td>304</td>
</tr>
<tr>
<td>PEGetMargins</td>
<td>305</td>
</tr>
<tr>
<td>PEGetNDetailCopies</td>
<td>306</td>
</tr>
<tr>
<td>PEGetNFormulas</td>
<td>307</td>
</tr>
<tr>
<td>PEGetNGroups</td>
<td>308</td>
</tr>
<tr>
<td>PEGetNGroupSortFields</td>
<td>309</td>
</tr>
<tr>
<td>PEGetNPages</td>
<td>309</td>
</tr>
<tr>
<td>PEGetNParameterCurrentRanges</td>
<td>310</td>
</tr>
<tr>
<td>PEGetNParameterCurrentValues</td>
<td>311</td>
</tr>
<tr>
<td>PEGetNParameterDefaultValues</td>
<td>312</td>
</tr>
<tr>
<td>PEGetNParameterFields</td>
<td>313</td>
</tr>
<tr>
<td>PEGetNSections</td>
<td>314</td>
</tr>
<tr>
<td>Function/Method</td>
<td>Page</td>
</tr>
<tr>
<td>----------------</td>
<td>------</td>
</tr>
<tr>
<td>PESetSQLExpression</td>
<td>430</td>
</tr>
<tr>
<td>PESetSQLQuery</td>
<td>432</td>
</tr>
<tr>
<td>PESetTrackCursorInfo</td>
<td>433</td>
</tr>
<tr>
<td>PESetWindowOptions</td>
<td>433</td>
</tr>
<tr>
<td>PEShow...Page</td>
<td>435</td>
</tr>
<tr>
<td>PEShowPrintControls</td>
<td>436</td>
</tr>
<tr>
<td>PEShowPrintControls</td>
<td>436</td>
</tr>
<tr>
<td>PEStartEventInfo structure</td>
<td>496</td>
</tr>
<tr>
<td>PEStartPrintJob</td>
<td>437</td>
</tr>
<tr>
<td>PEStopEventInfo structure</td>
<td>497</td>
</tr>
<tr>
<td>PESubreportInfo structure</td>
<td>498</td>
</tr>
<tr>
<td>PETableDifferenceInfo structure</td>
<td>500</td>
</tr>
<tr>
<td>PETableLocation structure</td>
<td>502</td>
</tr>
<tr>
<td>PETableType structure</td>
<td>504</td>
</tr>
<tr>
<td>PETestNthTableConnectivity</td>
<td>439</td>
</tr>
<tr>
<td>PETrackCursorInfo structure</td>
<td>506</td>
</tr>
<tr>
<td>PEValueInfo structure</td>
<td>508</td>
</tr>
<tr>
<td>PEVerifyDatabase</td>
<td>439</td>
</tr>
<tr>
<td>PEWindowOptions structure</td>
<td>511</td>
</tr>
<tr>
<td>PEZoomLevelChangingEventInfo structure</td>
<td>512</td>
</tr>
<tr>
<td>PEOpenWindowOptions method</td>
<td>55</td>
</tr>
<tr>
<td>Report Designer Database Object</td>
<td>114</td>
</tr>
<tr>
<td>ViewChanged Event</td>
<td>587</td>
</tr>
<tr>
<td>Report Viewer CRViewer Object</td>
<td>587</td>
</tr>
<tr>
<td>ViewReport method</td>
<td>576</td>
</tr>
<tr>
<td>Report Viewer CRViewer Object</td>
<td>576</td>
</tr>
<tr>
<td>ActiveX Control, upgrading from Crystal Custom Control</td>
<td>48</td>
</tr>
<tr>
<td>ActiveX Controls</td>
<td>46</td>
</tr>
<tr>
<td>adding ActiveX Control to project</td>
<td>46</td>
</tr>
<tr>
<td>adding Automation Server to project</td>
<td>49</td>
</tr>
<tr>
<td>adding Crystal Smart Viewer to project</td>
<td>557</td>
</tr>
<tr>
<td>changing properties for ActiveX Control</td>
<td>48</td>
</tr>
<tr>
<td>changing properties for ActiveX Control at runtime</td>
<td>48</td>
</tr>
<tr>
<td>creating User-Defined Functions, See User-Defined Functions.</td>
<td>48</td>
</tr>
<tr>
<td>dates and date ranges</td>
<td>43</td>
</tr>
<tr>
<td>embedded quotes in VB calls</td>
<td>42</td>
</tr>
<tr>
<td>hard-coded nulls in VB user defined types</td>
<td>45</td>
</tr>
<tr>
<td>releasing objects</td>
<td>51</td>
</tr>
<tr>
<td>sample Automation Server application</td>
<td>55</td>
</tr>
<tr>
<td>section codes and</td>
<td>26</td>
</tr>
<tr>
<td>solutions</td>
<td>41</td>
</tr>
<tr>
<td>string issues in VB links</td>
<td>44</td>
</tr>
<tr>
<td>using ActiveX Control</td>
<td>47</td>
</tr>
<tr>
<td>using Automation Server</td>
<td>50</td>
</tr>
<tr>
<td>using the Crystal Report Engine API in</td>
<td>42</td>
</tr>
<tr>
<td>Wrapper DLL</td>
<td>45</td>
</tr>
<tr>
<td>Visual Basic applications</td>
<td>42</td>
</tr>
<tr>
<td>when to open Crystal Report Engine</td>
<td>42</td>
</tr>
</tbody>
</table>

**W**

<table>
<thead>
<tr>
<th>Function/Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Reports Server</td>
<td>562</td>
</tr>
<tr>
<td>connecting to Crystal Smart Viewer</td>
<td>653</td>
</tr>
<tr>
<td>web site</td>
<td>653</td>
</tr>
<tr>
<td>Windows</td>
<td>653</td>
</tr>
<tr>
<td>see Microsoft Windows</td>
<td>653</td>
</tr>
<tr>
<td>working</td>
<td>653</td>
</tr>
<tr>
<td>with section codes</td>
<td>22</td>
</tr>
</tbody>
</table>

**Z**

<table>
<thead>
<tr>
<th>Function/Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zoom method</td>
<td>577</td>
</tr>
<tr>
<td>Report Viewer CRViewer Object</td>
<td>577</td>
</tr>
<tr>
<td>ZoomLevelChanged Event</td>
<td>588</td>
</tr>
<tr>
<td>Report Viewer CRViewer Object</td>
<td>588</td>
</tr>
</tbody>
</table>