R&R Report Writer
For Windows, Xbase Edition
Developing Applications
Liveware Publishing

Version 8.1

COPYRIGHT

© 2000 Liveware Publishing Inc.

All rights reserved.

Liveware Publishing Inc.
2-355 Clock Tower Place
Maynard, MA 01754

This manual is copyrighted and all rights are reserved. This document may not, in whole or part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without the prior written consent of Liveware Publishing Inc.

Printed in the United States of America

Trademarks and Acknowledgments

R&R Report Writer is a trademark of
Liveware Publishing Inc.

Portions of the imaging technology of this product are copyrighted by Accusoft Corporation.

All Avery product code numbers are trademarks of the Avery Dennison Corporation.

All other product names and logos in this manual are used for identification purposes only and may be trademarks or registered trademarks of their respective companies.

Documentation Part Number: RRW8MA(2000-01)

Table of Contents

Chapter 1 Overview 1

Introduction 1
Organization of the Manual 1
Runtime Requirements 2

Chapter 2 Using the Viewer Executable 3

Introduction 3
Executing the Viewer 3
Providing Viewer Input 4
The Viewer Command Line 4
Command Switches 5
Using the Viewer on a Network 9
Using RRW.INI for Default Information 10
Using Control Tables and Files 11
Creating a Control Table 11
Creating a Text Control File 11
Specifying Control Parameters 12
Parameters for Modifying Report Characteristics 13
Parameters to Control Viewer Preview Display 31
Understanding the Viewer Status File 33
Status File Fields 33
Application Calls to the Viewer Executable 36
Calling the Viewer from C 36
Calling the Viewer from Visual Basic 37
Calling the Viewer from Windows FoxPro 37
Calling the Viewer from PowerBuilder 38
Parameter Passing 38
Passing Control Parameter Values 38
Using a Parameter Table 41

Chapter 3 Accessing the Viewer DLL 43

Introduction 43
Action Routines 43
Get-Parameter Routines 44
Set-Parameter Routines 46

User-Interface Routines 48
Error-Handling Routines 49
Important Note 49
Functions Provided by the Viewer DLL 49
choosePrinter 50
chooseReport 50
endReport 52
execRuntime 52
getBeginPage 54
getCopies 55
getDisplayErrors 55
getDisplayStatus 56
getEndPage 56
getErrorInfo 57
getExportDest 59
getFilter 59
getFilterUsage 60
getFirstFieldName 60
getFirstFilteredFieldName 61
getFirstGroupField 62
getFirstRelationInfo 63
getFirstSortField 64
getFirstUserParam 65
getHighScope 66
getLibrary 66
getLowScope 67
getMasterIndexInfo 68
getMasterTableName 68
getMemoName 69
getNewReportHandle 70
getNextFieldName 70
getNextFilteredFieldName 71
getNextGroupField 71
getNextRelationInfo 72
getNextSortField 73
getNextUserParam 74
getOutputDest 74
getOutputFile 75
getPreventEscape 75
getPrinter 76
getPrinterPort 77

getReportPick 77
getRuntimeRecord 78
getScopeUsage 79
getStatusEveryPage 79
getTestPattern 80
getWinTitle 80
resetErrorInfo 81
setBeginPage 81
setCopies 82
setDataDir 82
setDisplayErrors 83
setDisplayStatus 84
setEndPage 84
setExportDest 85
setFilter 85
setFilterUsage 86
setGroupField 88
setHighScope 88
setImageDir 89
setIndexExtension 89
setLibrary 90
setLibraryDir 91
setLowScope 92
setMasterIndexInfo 92
setMasterTableName 93
setMemoName 94
setOutputDest 95
setOutputFile 96
setPreventEscape 97
setPrinter 98
setPrinterPort 99
setRelationInfo 100
setReportPick 101
setScopeUsage 102
setSortField 103
setStatusEveryPage 104
setStatusFileName 105
setSuppressTitle 105
setTestPattern 106
setUserParam 106
setWinBorderStyle 109
setWinControlBox 109

setWinHeight 110
setWinLeft 111
setWinMaxButton 111
setWinMinButton 112
setWinTitle 112
setWinTop 113
setWinWidth 114
setWriteAllow 114
setXbaseEditor 115
writeRuntimeRecord 115

Chapter 4 Using the Custom Control 117

Introduction 117
Installation 117
Determining Report Status 118
Using RRW.INI for Default Information 119
Using the Custom Control 119
Changing Values Using the Properties List 120
Changing Values Using the Control Properties Dialog . 121
Custom Control Properties 129
(About) 129
Action 129
CopiesToPrinter 130
DataDirectory 130
Destination 131
DisplayError 133
DisplayStatus 134
EndPage 134
ExportDestination 135
Filter 136
GroupFields 138
GroupFieldsString 139
HighScope 140
ImageDirectory 140
IndexExtension 141
LastErrorCode 142
LastErrorPage 143
LastErrorString 143
LoadProperties 144
LowScope 145
MasterIndex 145

MasterTable 147
MemoFileName 149
NoEscape 150
Parameters 150
ParametersString 151
Port 152
Printer 153
PrintFileName 154
Query 155
RelatedTables 157
RelatedTablesString 157
ReportDirectory 158
ReportLibrary 159
ReportName 160
ReportPick 161
ResetControl 162
ResetProperties 163
RunReport 163
Scope 164
SortFields 166
SortFieldsString 167
StartPage 168
StatusFileName 168
SuppressTitle 169
TestPattern 170
UpdateControl 170
WindowBorderStyle 171
WindowControlBox 172
WindowHeight 172
WindowLeft 173
WindowMaxButton 174
WindowMinButton 174
WindowTitle 175
WindowTop 176
WindowWidth 176
WriteAllow 177
XbaseEditor 178

Chapter 5 R&R Open Scripting 179

Introduction 179
Custom Report Wizards 179

Configuring the Custom Application 180
Invoking the Custom Application 180
Script File Format 181
Script File Sections and Keywords 183
Sample Script Output 191
Script Command-Line Argument (/S) 193
Report Wizard Input File 193

Chapter 6 Interfacing to Application DLLs 195

Introduction 195
Syntax 195
Example 196

Chapter 7 Distributing Reports 197

Introduction 197
Distribution Files for the Viewer Executable 197
Distribution Files for the Viewer DLL 199
Distribution Files for the OCX 200
Specifying the Location of RRWRUN.EXE 200
Distributing VB Applications 201
Retrieving Report Files 201
Master Table 202
Related Tables and Indexes 202
Text Memo Files 203
Image Files 203
Consistency Checking 204
Deleted Field 204
Deleted Linking Field 205
Changed Field Name or Data Type 205
Changed Field Width 206
Changed File Name 206
Save the Revised Report 206

Appendix A Viewer Equivalencies 207

Introduction 207
Table of Viewer Equivalencies 207

Index 209

Chapter 1
Overview

Introduction

This manual explains how to incorporate reports into your Windows applications, whether you are using the Viewer or distributing reports for use in interactive Report Designer. Using the Viewer, you can call reports from within an application program just as you might call any other program module.

For example, you might develop an Order Entry application that calls the Viewer to produce order forms, mailing labels, and invoices designed with Report Designer. Users can then access these forms and reports from Windows or from a Windows application without using interactive Report Designer.

Organization of the Manual

Chapters 2, 3, and 4 of this manual explain three methods for accessing the Viewer:

- You can directly access the Viewer executable (RRWRUN.EXE) using a control table or file. This method is explained in Chapter 2, "Using the Viewer Executable."
- The Viewer DLL provides an Application Programming Interface (API) that is suitable for use by any high-level programming language. See Chapter 3, "Accessing the Viewer DLL," for details.
- The R&R ActiveX control (OCX) simplifies Viewer access for Visual Basic applications. Chapter 4, "Using the ActiveX Control," explains this method.

The remaining chapters in this manual provide information for application developers who are creating reports for use in the Windows environment, whether the reports will be run via the Viewer or interactively:

- Chapter 5, "R&R Open Scripting," explains how developers can pass a user-specified report specification to interactive Report Designer by means of a script file.
- Chapter 6, "Interfacing to Application DLLs," explains use of R&R's CDLL( ) function to call a Windows Dynamic-Link Library (DLL) function from a report.
- Chapter 7, "Distributing Reports," provides information useful to application developers who are creating reports for distribution to other users.
- Appendix A, "Runtime Equivalencies," shows the equivalencies among the Custom Control properties, DLL routines, and Viewer executable control parameters, as well as the default value for each where applicable.

Runtime Requirements

To run the Viewer, you need the following:

- The Viewer program (RRWRUN.EXE), which is installed in the program directory if you choose to install the optional Viewer files during Setup.
- All Viewer distribution files required for your particular application. See Chapter 7, "Distributing Reports," for a complete list of required and optional files.
- A minimum of 500 KB of available memory for execution.

Chapter 2
Using the Viewer Executable

Introduction

This chapter explains how to use the Viewer executable (RRWRUN.EXE) to run reports from the Windows Program Manager or from within Windows application programs. The explanation of the Viewer executable is presented in the following sections:

- ❑ Executing the Viewer
- ❑ Using Control Tables and Files
- ❑ Understanding the Viewer Status File
- ❑ Application Calls to the Viewer Executable
- ❑ Parameter Passing

As noted in Chapter 1, using the Viewer executable is one of three available methods for accessing the Viewer. The other methods are explained in Chapter 3, "Accessing the Viewer DLL," and Chapter 4, "Using the ActiveX® Control."

Executing the Viewer

To use the Viewer executable to run a report, follow these general steps:

1. In interactive Report Writer, create and save each report you want to run.
2. Using your database software or a text editor, create a Viewer control table or file that identifies the report to be run, as well as any parameters you want to cha at runtime. The structure and contents of control tables and files are described in the **Using Control Tables and Files** section of this chapter.
3. Execute the Viewer in one of the following ways:
   - ♦ Click the Windows Start button and select Run; then enter the Viewer command line and select OK.

- Create a shortcut on the Windows desktop: right-click on an empty area of the desktop, highlight New, and select Shortcut. Enter the Viewer command line and select Next; enter a shortcut name and select Finish.
- Use the R&R Report Shortcut Maker utility to create shortcuts for your reports so that you can access them simply by double-clicking an icon.
- Include a call to the Viewer in your Windows application.

Providing Viewer Input

When you run a report with the Viewer executable, you use either a database table (referred to as a *control table*) or a text file (referred to as a *text control file*) to specify each report you want to run and any parameters you want to modify at runtime.

You create a control table using your database software. You can create a text control file using any database language, text editor, or word processor that produces unformatted text files. For details on control tables and files, see the **Using Control Tables and Files** section of this chapter.

The Viewer Command Line

After creating a report and a control table or file, you can use the Viewer to generate the report. When you call the Viewer, you must include the name of your control table or file and, optionally, one or more control table record numbers that identify the reports to be run.

Command Line Using a Control Table

If you use a database table to provide Viewer control parameters, the command syntax is:

```
RRWRUN <table name> [<record #> ...][switches]
```

Substitute the name (and optionally the path) of the control table for **<table name>**. For **<record #>**, substitute the record number(s) of the report(s) you want to run; if you do not include one or more record numbers, Viewer will execute all reports specified in the control table. In addition, you can optionally include one or more of the command switches listed in Figure 2.1.

For example, to generate the reports specified by records 1 and 2 of the RRWRUNIN.DBF control table, use a command like this:

```
RRWRUN RRWRUNIN  1  2
```

Command Line Using a Text Control File

If you use a text file to supply control parameters, the command syntax for running a single report is:

```
RRWRUN /T<file name> [switches]
```

Substitute the name (and optionally the path) of the text control file for **<file name>**. In addition, you can optionally include one or more of the command switches listed in Figure 2.1.

To run multiple reports using a text file, you must create a *command file*, an unformatted text file that lists the relevant text control files. First create a separate control file for each report you want to run; then create a command file listing the control files. To execute the Viewer with a command file, use the following syntax:

```
RRWRUN @<command file name> [switches]
```

For example, if you created three control files, you could then create a command file named REPORTS.CMD that lists these three control files (each on a separate line) and execute Viewer using the following command:

```
RRWRUN @REPORTS.CMD
```

You can optionally include one or more of the command switches listed in Figure 2.1.

Command Switches

The Viewer command can include one or more of the optional command switches listed in Figure 2.1. If you are using a text control file, the /T argument must appear first; the other switches can be in any order on the command line and can be either upper or lower case.

Switch	Specifies...
/T	Path and name of text control file
/R	Default report directory
/D	Default data directory
/I	Default image file directory
/X*nnn*	Default index file name extension
/W[Y|N]	Whether database users can (Y) or cannot (N) modify database files while in use by the Viewer
/O	Status file name and/or directory
/E[X|O]	Memo field type: (X) for an Xbase editor or (O) for another memo editor
/H	Suppression of Title and Summary lines when no records are found
/B	Suppression of product "splash screen" display at startup
/AL	Name of DLL to be pre-loaded

Figure 2.1 Command Switches

These switches are explained in the following sections.

Text Control File Name (/T)

To execute Viewer using a text control file, include the name (and optionally the path) of the control file preceded by /T as the first argument to the Viewer command.

For example, the following command will run the report specified in the SALES.TXT control file:

```
RRWRUN  /TSALES.TXT
```

Default Report Directory (/R)

To specify a default directory where the Viewer will look for the report or library specified in the RI_REPORT or RI_LIBRARY control table parameter, use the /R switch in the Viewer command. The directory you specify with this switch will override any default report directory specified in the RRW.INI file.

For example, the following command specifies C:\LIB as the default report directory:

```
RRWRUN  RRWRUNIN  1  2  /RC:\LIB
```

This command will run the reports specified in the first two records of the RRWRUNIN table. The Viewer will look for the report file in C:\LIB.

Default Data Directory (/D)

The Viewer looks for the data files (tables, indexes, and text memo files) used by a report in the directory saved with that report. To specify a default data directory where the Viewer will look for data files when they are not in the saved directory, use the /D switch in the Viewer command. The directory you specify with this switch will override any default data directory specified in the RRW.INI file.

For example, the following command specifies C:\DATA as the default data directory:

```
RRWRUN  RRWRUNIN  1  2  /DC:\DATA
```

This command will run the reports specified in the first two records of the RRWRUNIN table. If the Viewer cannot find the data files in the location saved with the report, it looks in C:\DATA.

Default Image File Directory (/I)

To specify a default directory where the Viewer may look for image files used in the report, use the /I switch with the Viewer command. The directory you specify with this switch will override any default image directory specified in the RRW.INI file.

For example, the following command specifies C:\IMAGES as the default image directory:

```
RRWRUN  RRWRUNIN  1  2  /IC:\IMAGES
```

Default Index File Extension (/X)

If the index files used with the Viewer have a different file extension than the index files saved with a report, you can use the **/X** switch to specify a default index file extension. The Viewer will look for index files with the default extension if it cannot locate the index files saved with the report. Syntax for this switch is **/X*nnn***, where ***nnn*** is the 1- to

3-character index file extension. The extension you specify with this switch will override any index file extension specified in RRW.INI.

For example, the following command specifies *ntx* as the file extension the Viewer should use when it cannot locate index files using their saved extensions:

```
RRWRUN  RRWRUNIN  1  2  /Xntx
```

File Write Access (/W)

When executing the Viewer in a multi-user or multi-tasking environment, you can use the **/W** switch to control whether database users can modify the tables and indexes in use by reports. Use the Y or N argument to the **/W** switch to control access to these files. The write access you specify with this switch will override any write access setting in RRW.INI.

To allow database users to modify files in use by a Viewer report, include the switch **/WY** in the Viewer command. This setting might not be appropriate for all reports, but it is useful when you want to allow database users to update files while you are reporting on them.

If you do *not* want users to modify files in use by a report, include the switch **/WN** in your Viewer command to prevent users from modifying tables and indexes while the report is running. Other R&R users will still have access to those files. (For more information about file write access settings, see the explanation of the **Allow Other Users to Update Database Tables** setting in Chapter 5, "Setting Defaults," in *Using R&R*.) Note that this switch controls R&R's behavior only when accessing shared data; your database or network software may impose other file access restrictions.

Status File Name (/O)

You can distinguish Viewer status files by using the **/O** switch to specify the directory in which the file will be created and/or to specify the complete status file name.

For example, the following command generates a status file named MYSTATUS. Because no path is specified, the status file will be created in the current directory.

```
RRWRUN  RRWRUNIN  5  /OMYSTATUS
```

To specify the directory in which a status file should be created, enter a full path and name. If you enter a path without a file name, the Viewer will create a table named RRUNOUT.DBF (if you are using a control table) or a text file named RRUNOUT.OUT (if you are using a text control file) in the specified directory.

Memo Editor (/E)

To identify the type of memo editor that was used to create database memos, use the **/E** switch. Use **/EX** to indicate that memos were created with an Xbase memo editor or **/EO** to indicate that memos were created with any other memo editor.

Title/Summary Lines for No Records Found (/H)

By default, the Viewer will print Title and Summary lines even when no records are found. To cause the Viewer to print nothing (no Title or Summary lines) when no records are found, execute the Viewer with the **/H** switch.

Suppress Splash Screen (/B)

By default, at startup the Viewer displays a "splash screen" containing product name and other information. To suppress display of this screen at startup, execute the Viewer with the **/B** switch.

Pre-Load DLL (/AL)

To specify the name of a DLL to be loaded at startup, use the **/AL** switch followed by the name (optionally including the path) of the DLL to be loaded.

Using the Viewer on a Network

Since the Viewer is available for unlimited use, you need not purchase more than one copy of R&R to enable multiple users to run Viewer reports. More than one user can run Viewer reports on a network with only a single copy of the Viewer programs stored on the network server. Use the Viewer commands in the previous section to run reports from within an application or from Windows.

Note that the Viewer control table is open for shared access when the Viewer program is active on a network. In some applications, it may be possible for two users to access the same record in the control table at

the same time. For example, two users might specify query expressions that affect the value of the RI_FILTER parameter in the control table. In this case, the Viewer would generate two reports using the second query expression. To better manage multiple-user access to the control table, create a separate record in the table for each user. You can also use the **/O** switch to create a unique output status table for each user.

Using RRW.INI for Default Information

If RRW.INI, the R&R configuration file, is in the Windows directory, the Viewer will use the default settings specified in that file. However, command-line switches take precedence; any setting you specify using a command-line switch will always override the corresponding RRW.INI setting.

Figure 2.2 lists the RRW.INI settings that the Viewer will use (unless a command-line switch is used instead).

RRW.INI Setting	*Specifies*
[Defaults] Section	
DataDir	Default data directory
ImgDir	Default image file directory
ImgExt	Default image file extension
LibDir	Default report/library directory
MemExt	Default memo file extension
IndExt	Default index file extension
AllowW	Database file write access
MemoEd	Default memo editor
[Preferences] Section	
PrevWinClr	Preview window color
ShowSplash	Suppression of product “splash screen” display at startup

Figure 2.2 RRW.INI Settings Used by Viewer

Using Control Tables and Files

The following sections describe the structure and contents of database control tables and text control files.

- Creating a Control Table
- Creating a Text Control File
- Specifying Control Parameters

Creating a Control Table

Use your database software to create a table that contains columns for the control parameters; then add a row for each report you want to run and enter values in the appropriate columns. In the simplest case, the table can include just the report name (as the RI_REPORT value); as a result, the report will be output to the destination saved with the report.

Follow these guidelines in creating a control table:

- Specify each parameter in a separate column; the column name must be the same as the parameter name.
- Predefined parameters must use the column names and data types specified in Figures 2.3 and 2.11. For character columns, use any supported character data type. For numeric parameters, use any numeric data type, but note that the numeric parameters accommodate only integer values.
- User-defined parameters can use any column name and supported character data type.
- Parameters can be in any order.
- The only required parameter is RI_REPORT (and, for reports from a library, RI_LIBRARY) or, alternatively, RI_REPPICK.
- You can omit unused (blank) parameters.
- Specify parameters for each report in a separate row.

Creating a Text Control File

You can also create a control file using any text editor or word processing program that saves unformatted text files. In the file, you specify the report name and any optional parameter values. You must create a separate text file for each report you want to run. However, you can run multiple reports with a single Viewer command by creating a *command file* that lists each control file.

The format for each parameter name and value in a control file is:

```
<parameter name>=<value>
```

Follow these guidelines in creating a text control file:

- Specify each parameter and its value on a separate line.
- Each parameter and its value must fit on a single line.
- The maximum length of a line is 1000 characters.
- You can list parameters in any order.
- Parameter names are case insensitive (that is, you can enter them in upper, lower, or mixed case).
- Predefined parameters must use the names listed in Figures 2.3 or 2.11; user-defined parameters can have any name. See the section in this chapter entitled **Parameter Passing** for information about user-defined parameters.
- The only required parameter is RI_REPORT (and, if running reports from a library, RI_LIBRARY) or, alternatively, RI_REPPICK.
- Leading and trailing white space in both the parameter name and the value is ignored.
- Lines beginning with a left square bracket ([ ) are ignored.
- Lines beginning with a semicolon are ignored.

Specifying Control Parameters

The control table or file can contain two kinds of parameters:

- Predefined parameters specifying values that control frequently changed report features such as queries. The control table *must* contain a column for RI_REPORT (or RI_REPPICK). It can contain values for some or all of the other predefined parameters. These parameters are listed in Figures 2.3 and 2.11.
- User-defined parameters specifying values that control other report features such as user-supplied text strings, for use by the RIPARAM( ) function in the report. The control file need not include any user-defined parameters. If any of these parameters are present, the control file can contain values for some or all of them. For information about adding columns to the control file for user-defined parameters, see the Parameter Passing section of this chapter.

If the control file does not contain a valid entry for report name (and report library, if applicable), the Viewer cannot run the report. If the control table does not contain values for any predefined or user-defined parameters designed to change characteristics of the saved report, the Viewer will run the report as saved.

Width of Predefined Parameters

Predefined parameters have maximum widths specified in Figure 2.3 and 2.11. While you should not exceed these widths, you can decrease the widths of these parameters to correspond to the actual width of your data. For example, if you plan to specify a scope using strings that will be no longer than 20 characters, you can define the RI_LOSCOPE and RI_HISCOPE parameters as 20 characters wide. User-defined parameters can be up to 512 characters wide.

Parameter Values

Parameters that require character values can contain upper, lower, or mixed case letters, unless the parameters contain values used in a query. By default R&R is case insensitive, but if you edited RRW.SRT to make R&R case sensitive, you should enter query values in the case used in the database.

Some predefined parameters can have a question mark (**?**) value in the control table or file. Use the question mark to specify that the Viewer should display a dialog box prompting the user to enter or select a value. For example, when RI_PRINTER contains a question mark, the Viewer will display a dialog box prompting the user to choose screen, printer, or export as the report's output destination.

The question mark parameter value is explained in more detail in the descriptions of the parameters for which it is valid: RI_PRINTER, RI_REPPICK, RI_WPTR, RI_WPORT, RI_SCOPE, RI_QUERY, and user-defined parameters.

Parameters for Modifying Report Characteristics

Figure 2.3 lists the predefined Viewer control parameters that can be used to control report characteristics. The next section of this chapter lists and explains the parameters that apply specifically to the size and appearance of the preview window at runtime.

In the Data Type column, the letter **C** represents the character data type, **N** represents numeric, and **L** represents logical.

Parameter Name	Contents	Data Type	Max. Width
RI_ALIAS1 – RI_ALIAS99	In each, a related table, index, or index tag name	C	300
RI_BEGPAGE	Beginning page number	N	9
RI_CHKTIME	Checkpoint frequency flag	C	1
RI_COPIES	Number of copies	N	9
RI_DISPERR	Display-error flag	C or L	1
RI_ENDPAGE	Ending page number	N	9
RI_EXPDST	Export destination	C	8
RI_FILTER	Query expression	C	1024
RI_GROUP1 – RI_GROUP8	Group field override	C	20
RI_HISCOPE	High scope value	C	250
RI_LIBRARY	Report library name	C	128
RI_LOSCOPE	Low scope value	C	250
RI_MASTER	Master table name	C	128
RI_MEMO	Text memo file name	C	128
RI_MINDEX	Master index information	C	150
RI_NOESC	User escape flag	C or L	1
RI_OUTFILE	Output file name	C	128
RI_PRINTER	Destination	C	32
RI_QUERY	Query flag	C	1
RI_REPORT	Report name	C	30
RI_REPPICK	Report select/pick flag	C	1
RI_SCOPE	Scope flag	C	1
RI_SORT1 – RI_SORT8	Sort field override	C	21
RI_STATUS	Display-status flag	C or L	1
RI_TEST	Test-pattern flag	C or L	1
RI_WPORT	Printer port	C	40
RI_WPTR	Printer name	C	40
RI_WTITLE	Window title	C	200

Figure 2.3 Predefined Viewer Control Parameters

RI_ALIAS1 – RI_ALIAS99

These parameters are optional. You can use each of the parameters to specify a related table, index, and/or index tag name to override those saved with the report. The syntax for a file override specification is:

```
<alias> = <table>,<index>,<tag>
```

In this specification, **<alias>** represents the alias assigned to the table in the saved report; **<table>** represents the replacement table, which can be a complete path and table name, a directory, or a table name; **<index>** represents the name and optional path of the new index file; and **<tag>** represents the name of an index tag if the specified index file is a multiple-field index file. You must specify at least one of the three replacement names in any override specification; you can specify all three.

For example, the following specification replaces the related table assigned the CUST95 alias in the saved report with a table named CUST96.DBF in the DATA directory on drive C. It also replaces the saved index file with the CUST96.MDX index file, and the saved index tag with the CUSTID index tag:

```
CUST95 = C:\DATA\CUST96,C:\DATA\CUST96.MDX,CUSTID
```

You can also use a file override specification to replace just the index file, just the index tag, or both. If you omit the <table> or <index> part of the specification, you must use a comma as a place holder.

For example, the following specification replaces the index file used to read the CUST96 table with the LASTNAME.NDX file in the DATA directory on drive C. The comma serves as a place holder for the <table> specifier:

```
CUST96 = ,C:\DATA\LASTNAME.NDX
```

For a multiple-field index, the following override specification replaces the saved tag used with the CUST96 table and index with the LASTNAME tag:

```
CUST96 = ,,LASTNAME
```

If you do not include file override specifications in any of these fields, the Viewer uses the database and index files saved with the report. It searches for these files using the search rules explained in Chapter 7, “Distributing Reports.”

RI_BEGPAGE, RI_ENDPAGE

These parameters are optional. The beginning and ending page number parameters allow you to override the starting and ending page numbers saved with the report. The default value for these parameters is blank.

To specify page numbers, include an RI_BEGPAGE value, an RI_ENDPAGE value, or both. If you specify both, RI_ENDPAGE must be equal to or greater than RI_BEGPAGE. For example, users can restart a canceled report where it was interrupted by specifying the starting page number as the RI_BEGPAGE value. (See the description of the RO_PAGES field in the **Understanding the Viewer Status File** section.) To reprint one or more consecutive pages of a report, specify the page numbers in the RI_BEGPAGE and RI_ENDPAGE parameters. To print just one page, specify the same page number for both parameters.

RI_CHKTIME

This parameter is optional. The checkpoint frequency flag controls how frequently the Viewer status table, by default RRUNOUT.DBF, is updated. The checkpoint flag can contain the letter **R** or **P**. **R** tells the Viewer to update the RRUNOUT table after completing each report; **P** tells the Viewer to update the table after completing each page. The default value is **R**.

Specify **P** as the checkpoint value if you want Viewer users to be able to determine how much of a report was printed before an abnormal termination (for example, a system failure). When this value is **P**, the Viewer will update the RO_PAGES page number value in the status table after each page of the report is processed. (See the section entitled **Understanding the Viewer Status File**.) In case of a report termination, the report can be restarted where it left off.

If your application doesn't require the ability to restart terminated reports, specify **R** and the report will print a bit faster. Users can always reprint a report starting at the beginning.

RI_COPIES

This parameter is optional. It contains the number of copies of the report you want to print. The number must be between 0 and 999, inclusive. If you leave this parameter blank or enter 0, the Viewer prints the number of copies saved with the report.

RI_DISPERR

This parameter is optional. It controls whether errors encountered by the Viewer are displayed on the screen. If the parameter contains a true logical value (**T**), any Viewer error messages are displayed in addition to being written to the Viewer status table, by default RRUNOUT.DBF. If the parameter contains a false logical value (**F**) or is blank, the Viewer error messages are not displayed, but are written to the Viewer status table. If the Viewer cannot open the status table, an error message is displayed regardless of the RI_DISPERR value.

RI_EXPDST

Use this parameter to specify the destination (display, file, or printer) for a report saved with an Export Type setting of Excel PivotTable or Excel Chart (you must also specify the appropriate value in RI_PRINTER). A value of **D** will cause Excel to display the PivotTable or Chart; **F** will cause Excel to send it to the file specified by RI_OUTFILE; and **P** will cause Excel to print it to its default printer.

RI_FILTER

The optional RI_FILTER parameter contains a logical expression that will override the query saved with a report, if any, when the value in RI_QUERY is **O** for Override.

RI_FILTER expressions use the same syntax as calculated field expressions that return logical values. (For details, see Chapter 7, "Working with Calculated Fields," in *Using R&R*.) The RI_FILTER expression can be up to 1024 characters long. When an expression is specified and the value of RI_QUERY is **O**, the Viewer selects all records where the value of the RI_FILTER expression is true. The expression can refer to any data available in the report, as well as many calculated and total fields.

For example, if you enter the expression **CITY="Dallas"**, the Viewer will select all records where the value of this expression is true, in other words all records where the value in the CITY field is Dallas. If the city name is in a memo field named NOTE, the expression **NOTE="*Dallas*"** will select all records in which the NOTE field contains the word "Dallas."

Entering the expression **PASTDUE=T** tells the Viewer to select all records where the value in the PASTDUE field is the logical true value. Entering **AMOUNT>=200** will select all records where the value in the AMOUNT field is equal to or greater than 200. Entering the following expression will select all records where the date in the INVDATE field of the RRORDERS table is January 31, 1996:

```
RRORDERS->INVDATE={01/31/96}
```

You can enter compound expressions using parentheses. For example, the following expression selects all records where the value in the CITY field is either Dallas or Houston and where the value in the SALES field is greater than 50,000:

```
(CITY="Dallas" or CITY="Houston") and SALES>50000
```

Certain restrictions apply to the use of total fields in RI_FILTER expressions. The totals must be pre-processed. If the totals are group totals, you can use only the highest-level preprocessed total in the RI_FILTER expression. For example, if the report has pre-processed subtotals at group levels 1 and 2, the RI_FILTER expression can contain only the level-1 pre-processed subtotal.

RI_GROUP1 – RI_GROUP8

The optional RI_GROUP parameters (RI_GROUP1 through RI_GROUP8) enable you to specify different group fields from those saved with the report. Figure 2.4 explains the possible values for these parameters (in each case, substitute the table alias for **alias** and the field name for **fieldname**).

Value	*Changes Group Selection to*
alias->fieldname	Field *fieldname* in table *alias*
fieldname	Field *fieldname* (*fieldname* must be unique)

Figure 2.4 Group Field Override Values

You must specify group overrides beginning with the first level you want to change and proceeding to the depth desired (that is, you cannot skip group levels).

RI_LIBRARY

This parameter is necessary only if you are running reports from a report library file; it identifies the library containing the report(s) you want to run. The library name can include a path. The .RP5 extension is optional. For example, a value of C:\DBASE\DATA\CUSTOMER in this parameter identifies the report library as CUSTOMER.RP5 in the subdirectory \DBASE\DATA on drive C.

If you don't include a path, the Viewer searches for the file in the default library directory specified on the command line or in RRW.INI. If no default is specified on the command line or in RRW.INI, the Viewer searches for the library in the current directory.

If the library you specify cannot be found or read, the Viewer writes an error in the status table and, optionally, displays an error message box (see RI_DISPERR).

RI_MASTER

This parameter is optional. It contains the name and/or directory location of a table that will override the master table saved with the report. The fields in the master table you specify with RI_MASTER must match in name, number, and data type the fields in the original master table.

- If you specify both a directory and a table name, this directory is the only directory searched and this table name is the only table the Viewer searches for.
- If you specify a directory without a table name, the Viewer searches the specified directory for the master table name saved with the report.
- If you specify a table name without a directory, the Viewer searches for a table with the specified name in the directory of the master table saved with the report, then in the default data directory specified on the command line or in RRW.INI. If no default data directory is specified, the Viewer searches for the table in the current directory.

If you omit this parameter or leave it blank, the Viewer uses the master table saved with the report.

RI_MEMO

This parameter is optional. It contains the name and/or directory location of the text memo file used in the report, which will override the text memo file saved with the report (if any).

- If you specify both a directory and a file name, this directory is the only directory searched and this file name is the only file the Viewer searches for.
- If you specify a directory without a file name, the Viewer searches the specified directory for the text memo file name saved with the report.
- If you specify a file name without a directory, the Viewer searches for a file with the specified name in the directory saved with the report, then in the default data directory specified on the command line or in RRW.INI.

If you leave this parameter blank, the Viewer uses the text memo file saved with the report, if any.

RI_MINDEX

The RI_MINDEX parameter enables you to specify a master index for a report that was saved without one, override the master index saved with a report, or remove a saved master index from a report. To specify a master index or override the saved one, the specification can consist of any or all of three values in the following order:

```
<index>,<index type>,<tag>
```

In this specification, **<index>** represents the index file specification, which can be a full path and file name, a directory, or a file name.

- If you specify both a directory and a file name, this directory is the only directory searched and this file name is the only file the Viewer searches for.
- If you specify a directory without a file name, the Viewer searches the specified directory for the master index name saved with the report.
- If you specify a file name without a directory, the Viewer searches for a file with the specified name in the directory of the master index saved with the report, then in the current master database directory, then in the default data directory

specified on the command line or in RRW.INI. If no default is specified, the Viewer searches in the current directory.

In this specification, **<index type>** is the data type of the new index, represented as **A** for datetime, **C** for character, **D** for date, or **N** for numeric. You can omit this value if you are overriding a saved master index and the replacement index is the same type (although it is good practice to include the data type specifier). If the index named in the specification is a multiple-field index file (MDX, CDX, or WDX), the **<tag>** part of the specification represents an index tag. If you omit the **<index>** or **<index type>** part of the specification, use a comma as a place holder for each.

For example, the following specification replaces the master index saved with the report with an index named CUST96.MDX in the CUSTOMER directory on drive C. Since the replacement index is the same data type as the original index, the specification includes a comma in place of the <index type>. It also replaces the saved index tag with the CUSTID index tag:

```
C:\CUSTOMER\CUST96.MDX,,CUSTID
```

The following specification changes only the index tag of the master index saved with the report:

```
,,CUSTID
```

To remove the master index saved with the report without specifying a new master index, use the RI_MINDEX value **,R**.

If you omit this parameter or leave it blank, the Viewer uses the master index saved with the report, if any.

RI_NOESC

This parameter is optional. The user escape flag can contain either a true (**T**) or false (**F**) logical value. True means the Cancel button in the status window is not active while reports are being output. False means the user can select Cancel during report output to pause or end the job. The default value is false. Note that the status window appears only when RI_STATUS is set to true.

If the user cancels the report, the RO_ECODE field in the status table contains a C (see **Understanding the Viewer Status File**).

RI_OUTFILE

This parameter is optional. It contains the name of an output file. Use it to send report output to a file, or use it in combination with RI_PRINTER and/or RI_EXPDST to export to any export type saved with the report. To send the report directly to the saved destination, omit this parameter or leave it blank.

- When RI_PRINTER is empty or contains the **D** or question mark (**?**) value, the Viewer outputs the report (including printer codes) to the file specified in RI_OUTFILE .
- When RI_PRINTER contains **A**, **X**, or **W**, the Viewer exports the report to the file specified in RI_OUTFILE (overriding the saved file name) as a text file (without printer codes), Xbase file, or worksheet file.
- When RI_PRINTER contains **CSV**, **MSWORD**, or **RTF**, the Viewer exports the report to the file specified in RI_OUTFILE (overriding the saved file name) as a text data (comma-, tab-, or character-separated) file, Word Merge file, or Rich Text Format file.
- When RI_PRINTER contains **Excel Chart** or **Excel PivotTable** and RI_EXPDST is **F** (for file), the Viewer exports the report to the Excel file specified in RI_OUTFILE.

The name of the output file can include a path. For example, to send a report to a text file INVOICE.TXT in the C:\PROJECT\TEXT subdirectory, specify the following value for the RI_OUTFILE parameter:

```
C:\PROJECT\TEXT\INVOICE.TXT
```

If RI_OUTFILE does not include a path, the Viewer places the file in the current directory.

RI_PRINTER

This parameter is optional and can have one of the following values: **D**, **A**, **P**, **Excel Chart**, **Excel PivotTable**, **RTF**, **CSV**, **MSWORD**, **W**, **X**, or a question mark (**?**).

The **D** value specifies that the report be sent to the display, allowing the user to preview the report before printing it. After previewing the report, the user can select Print on the Preview screen to send the report to the printer saved with the report or specified as the RI_WPTR

value. Note that if the value of RI_PRINTER is **D** and RI_OUTFILE is specified, the report will be output to the file specified in RI_OUTFILE when the user selects Print in the Preview screen.

The **A** value specifies that the report be sent to the text file named as the RI_OUTFILE value. The Viewer will export the report as a text file without printer codes.

The **P** value specifies that the report be sent to the printer saved with the report or specified as the RI_WPTR value, even if the report's saved destination is a file.

The **Excel Chart** and **Excel PivotTable** values specify that the report be exported to an Excel Chart or Excel PivotTable, respectively. If you specify one of these values, you can also include a value for RI_EXPDST to control the output destination (display, file, or printer).

Use the **CSV**, **MSWORD**, or **RTF** value to export to a text data file, Word Merge file, or Rich Text File, respectively. You can also specify an RI_OUTFILE value to override the output file name saved with the report.

To output a report to a worksheet or Xbase file, specify **W** or **X**, respectively, as the RI_PRINTER value and specify the file name as the RI_OUTFILE value. If you do not specify a file extension, R&R appends .WKS to worksheet files and .DBF to Xbase files.

The question mark (**?**) value allows the user to select the print destination (screen, printer, or export) at runtime. When the value of RI_PRINTER is a question mark, the user will see the dialog box shown in Figure 2.5. If RI_WTITLE is specified, the title bar will contain the RI_WTITLE value. If RI_WTITLE is empty, the title bar will contain the report name.

The user can select Screen to preview the report, Printer to print it, or Export to export it. If RI_OUTFILE contains a file name, the report will be output to the file specified by the RI_OUTFILE value if the user selects Export.

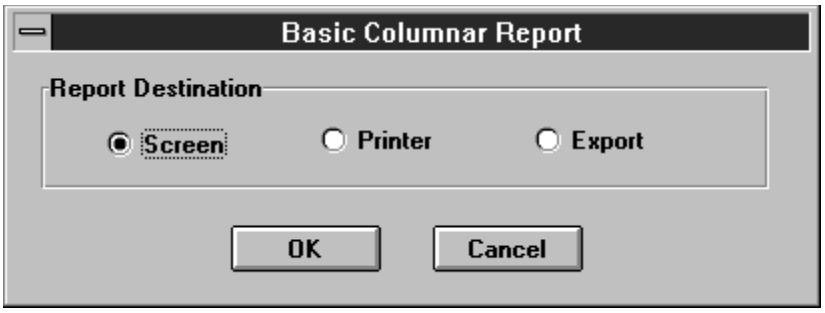

Figure 2.5 Print Destination Dialog Box

If you omit this parameter or leave it blank and RI_OUTFILE is empty or missing, the Viewer outputs the report to the printer saved with the report or specified as the RI_WPTR value. If you omit this parameter or leave it blank and RI_OUTFILE contains a file name, the Viewer outputs the report to a file with printer codes.

RI_QUERY

The optional RI_QUERY parameter allows you to control whether a query is applied to the report. RI_QUERY can have one of four values:

- **S** (Saved) means to run the report using the query saved with it, if any. The Viewer will ignore the expression in RI_FILTER and run the report exactly as it was saved.
- **E** (Entire) means to ignore any query saved in the report or contained in the RI_FILTER parameter.
- **O** (Override) means to override the saved query (if any) with the expression in the RI_FILTER parameter. The Viewer will generate the report with the records selected by the RI_FILTER expression.
- **?** (Question mark) means to allow the user to enter a query or edit the saved query at runtime. If no query was saved with the report, the Insert Selection Rule dialog displays, as shown in Figure 2.6.

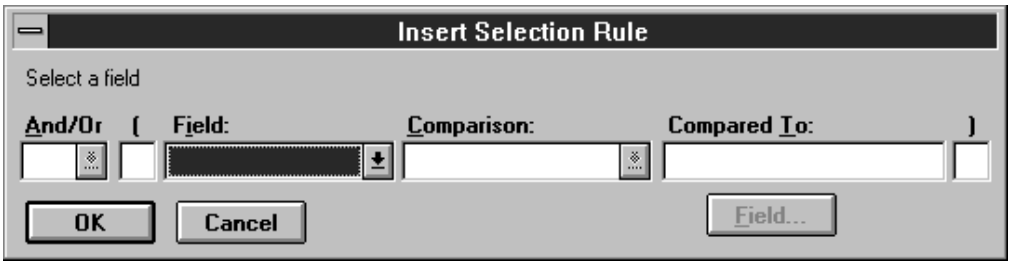

Figure 2.6 Insert Selection Rule Dialog Box

If a query was saved with the report, the Query dialog box displays, as shown in Figure 2.7.

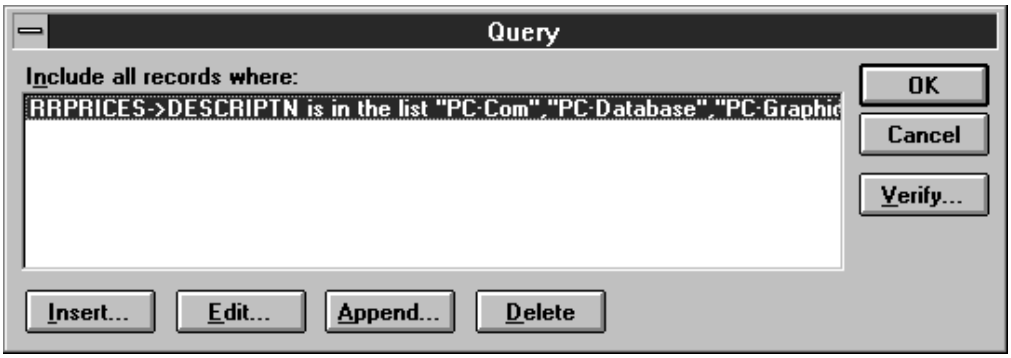

Figure 2.7 Query Dialog Box

When RI_QUERY contains a question mark (**?**), the value of RI_FILTER is always ignored.

RI_REPORT

This parameter is required (unless a value of **R** or **?** has been supplied for RI_REPPICK). It contains the name under which the report was saved. For example, to run a report named “Order Invoice” that was saved as a compound document file, enter **Order Invoice** in this parameter (note that you do not need to include the RRW extension). Except for case, you must enter the name exactly as it was saved.

If the report you want to run is in a report library, you must also include the appropriate value for RI_LIBRARY. The report must be in the library specified by RI_LIBRARY.

If you leave this parameter blank or if the report you select cannot be retrieved, the Viewer writes an error in the status table and, optionally, displays an error message box (see RI_DISPERR).

RI_REPPICK

This parameter is optional and can contain one of two values: **?** or **R**. If you include this parameter, you do not need to include the RI_REPORT parameter; if you include both RI_REPPICK and RI_REPORT values, Viewer ignores the RI_REPORT value.

Use the question mark (**?**) value in this parameter to have the Viewer prompt the user to select a succession of reports. When the value is a question mark (**?**), Viewer will prompt the user to select a report. After Viewer executes the selected report, the user will then be prompted to select another report. This prompt for report selection will repeat after each report until the user selects Cancel.

Use the **R** value in this parameter to prompt the user to select just one report. When the value is **R**, Viewer will prompt the user to select a report (as with the **?** value), but will not prompt for an additional report selection after the report has been executed.

RI_SCOPE, RI_LOSCOPE, RI_HISCOPE

These parameters are optional. The scope flag, RI_SCOPE, allows you to control the range of master table records that should be included in the report. You can specify a range of record numbers or index key values, ignore the scope saved with a report, or prompt the user to enter a range at runtime. When you specify scope values, the Viewer reads only the records in the master table whose record number or index key is within the specified range. You can often speed up a report by using scope values with a master index. If you omit this parameter or leave it blank, the Viewer uses the saved scope values.

RI_SCOPE can contain one of four values:

- **S** for "Saved," which means to use the scope values saved with the report.
- **E** for "Entire," which means to ignore any scope values.
- **O** for "Override," which means to override the saved scope values with the values in the two parameters RI_LOSCOPE and RI_HISCOPE. (Be careful to use the letter **O** and not the digit zero, 0.)

- Question mark (**?**), which allows the user to enter or change scope values at runtime. When RI_SCOPE contains a question mark, the dialog box shown in Figure 2.8 displays. If RI_WTITLE is specified, the title bar will contain the RI_WTITLE value. If RI_WTITLE is blank or missing, the title bar will contain the report name.

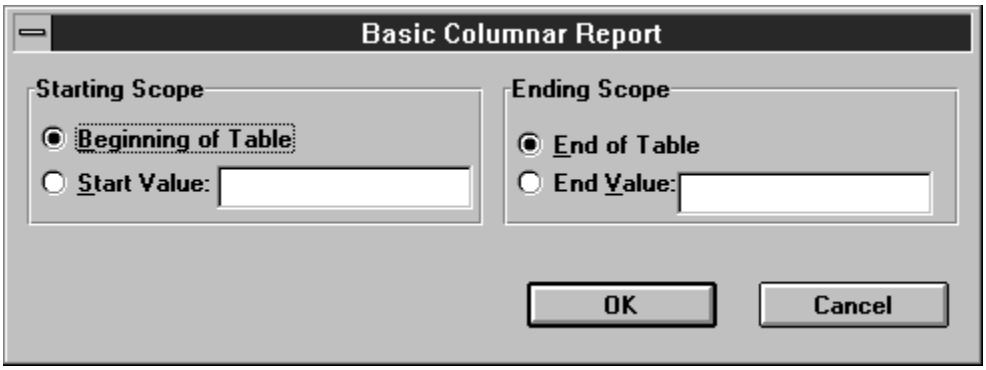

Figure 2.8 High and Low Scope Dialog Box

If RI_SCOPE contains **O** (Override), RI_LOSCOPE specifies the starting value of the scope and RI_HISCOPE specifies the ending value of the scope. If RI_SCOPE contains **S**, **E**, or **?**, the Viewer ignores RI_LOSCOPE and RI_HISCOPE.

Each RI_LOSCOPE and RI_HISCOPE value can contain either a record number or an index key value up to 250 characters wide. If no master index was saved with the report (or added using RI_MINDEX), the Viewer assumes the value is a record number. Otherwise, the Viewer assumes the value is a key value in the master index. In this case, the report begins reading the master table at the first record equal to or greater than the RI_LOSCOPE value and stops reading the master table after the last record found that is equal to or less than the RI_HISCOPE value.

The range fully includes the end points. In other words, if you enter A as the low value and M as the high value, the Viewer reads the first record in which the value begins with A through the last record in which the value begins with M. For example, if you have a customer table indexed on last name and you want to print invoices for all customers whose name begins with a letter between A and M, enter O in RI_SCOPE, the letter A in RI_LOSCOPE, and the letter M in RI_HISCOPE.

All scope values must be character strings. Note that a date scope value must be in the format mm/dd/yy or mm/dd/yyyy. Do not enclose scope values within quotes.

RI_SORT1 – RI_SORT8

The optional RI_SORT parameters (RI_SORT1 through RI_SORT8) enable you to specify different sort fields from those saved with the report. Figure 2.9 explains the possible values for these parameters (in each case, substitute the table alias for **alias** and the field name for **fieldname**).

Value	*Changes Sort to*
+alias->fieldname	Field *fieldname* in table *alias*, ascending
-alias->fieldname	Field *fieldname* in table *alias*, descending
alias->fieldname	Field *fieldname* in table *alias*, ascending
+fieldname	Field *fieldname*, ascending (*fieldname* must be unique)
-fieldname	Field *fieldname*, descending (*fieldname* must be unique)

Figure 2.9 Values for RI_SORT Parameters

You must specify sort overrides beginning with the outermost sort field and proceeding to the last level you want to override (that is, you cannot skip sort levels).

RI_STATUS

The RI_STATUS parameter enables you to specify whether the Viewer should display a status window while it is generating a report. If the parameter contains a true logical value (**T**), the Viewer will display a status window. If RI_STATUS is set to true and RI_NOESC is set to false, the status window will contain a Cancel choice that allows the user to terminate a report in progress.

If the RI_STATUS parameter is missing, empty, or contains a false logical value (**F**), the Viewer will not display a status window; instead it will display as an icon while it is running.

RI_TEST

This parameter is optional. The test pattern flag can contain either a true (**T**) or false (**F**) logical value. True means to display a dialog box before printing the report to allow the user to print a test pattern. False (or blank) means don't offer the user the choice to print a test pattern.

A test pattern is useful for aligning forms in the printer. The user can print the test pattern as many times as necessary and then print the report. If you enter **T**, the Viewer displays a box containing OK, Cancel, and Print buttons. The user can select OK and print as many test patterns as necessary to align the forms. Once the forms are aligned, the user can select Print to begin printing the actual report.

Note that a test pattern includes only page header, record, and page footer lines.

RI_WPORT

This parameter is optional. Enter a value such as **LPT1:** to override the printer port (and the printer associated with that port) saved with the report. Note that the colon is required. If both RI_WPTR and RI_WPORT values are supplied, they must match an installed Windows printer.

You can also use the question mark (**?**) value or enter the word **Default** for this parameter. When RI_WPORT contains a question mark, the user will see the Print dialog box shown in Figure 2.10. When RI_WPORT contains **Default**, Viewer will use the default Windows printer and port. See the description of the RI_WPTR parameter.

RI_WPTR

This parameter is optional. Enter one of the following values to override the printer saved with the report:

- The name of an available Windows printer (for example, "HP LaserJet Series III"). The value is case insensitive. If you enter a value in this parameter and RI_WPORT is blank, Viewer uses the port associated with the printer name in the list of available Windows printers.

- The question mark (**?**) value, to allow the user to select a printer at runtime. When RI_WPTR contains a question mark, the Print dialog displays, as shown in Figure 2.10.
- The word "Default" to force the Viewer to use the current default Windows printer. Use this setting only if you are sure that the default printer is compatible with the layout of your Viewer report(s)

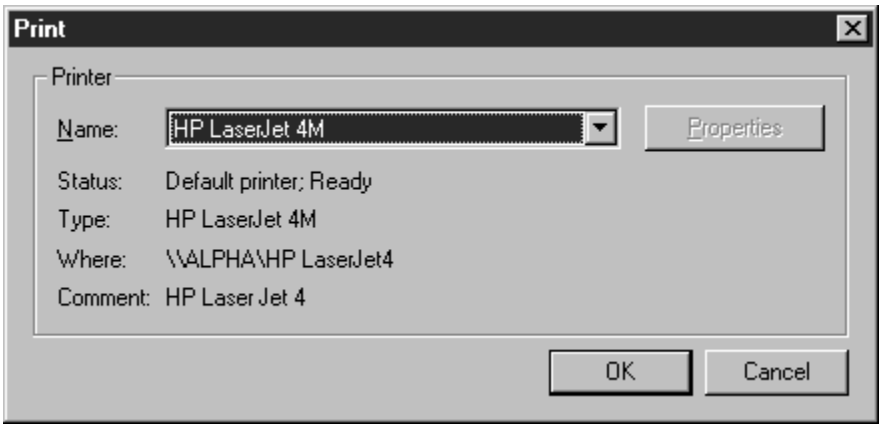

Figure 2.10 Print Dialog Box

The Printers applet (accessible from the Windows Control Panel) controls which printers are listed in the Print dialog box. Initially, the printer saved with the report is highlighted. The user can select another printer and port as necessary.

If this parameter is blank, the Viewer uses the printer saved with the report. If the report was saved with the "Print to File" option selected and the value of RI_PRINTER is blank, the RI_WPTR value controls which printer driver the Viewer will use. If the value of RI_PRINTER is **A**, **CSV**, **MSWORD**, **RTF**, **W**, or **X** for export to a file, the Viewer will ignore the value in RI_WPTR.

RI_WTITLE

This parameter is optional. Use this parameter to specify a report title (for example, "Quarterly Profits") that will display in the following places:

- The Title Bar of the Preview window;
- The Print Status window (if RI_STATUS = T);

- Below the Viewer icon (if RI_STATUS = F);
- The title bar of the dialog box that displays when a question mark is specified as the value for RI_REPPICK, RI_PRINTER, RI_SCOPE, or any user-defined parameter.

If this parameter is blank, the Viewer will use the report name as the window title.

Parameters to Control Viewer Preview Display

The parameters listed in Figure 2.11 control the size and appearance of the preview window at runtime.

Parameter Name	Controls	Data Type	Max. Width
RI_WBORDER	Presence and type of border	N	1
RI_WCTRL	Presence of system control box in caption bar	L	1
RI_WHEIGHT	Height of preview window in pixels	N	4
RI_WLEFT	Left position of preview window in pixels	N	4
RI_WMAX	Presence of maximize button in caption bar	L	1
RI_WMIN	Presence of minimize button in caption bar	L	1
RI_WTOP	Top position of preview window in pixels	N	4
RI_WWIDTH	Width of preview window in pixels	N	4

Figure 2.11 Control Parameters for Viewer Report Preview

The following sections explain the purpose and valid entries for each parameter.

RI_WBORDER

This parameter enables you to control whether the Viewer preview window is fixed or sizable. You can enter one of the following numeric values:

- **1** results in a fixed-size preview window with a standard border.

- **2** results in a variable-size preview window with a standard border.

RI_WCTRL

Use this parameter to specify whether the preview window will have a system control box (for switching to other applications or for closing the preview window) in the caption bar. If this parameter contains a logical **True** value, the window will have a control box; if **False**, the window will not have a control box.

RI_WHEIGHT

This parameter controls the height of the preview window. Enter the height value in screen pixels. Note that in order to control the size of the preview window at runtime, you must enter values for *both* RI_WHEIGHT and RI_WWIDTH.

RI_WLEFT

This parameter controls where the left edge of the preview window will be anchored. Enter the position in screen pixels.

RI_WMAX

Use this parameter to control whether the preview window will have a maximize control in the caption bar at runtime so that a user can run a report full-screen. If this parameter contains a logical **True** value, the window will have a maximize control; if **False**, the window will not have a maximize control.

RI_WMIN

Use this parameter to control whether the preview window will have a minimize control in the caption bar at runtime. If this parameter contains a logical **True** value, the window will have a minimize control; if **False**, the window will not have a minimize control.

RI_WTOP

This parameter controls where the top of the preview window will be anchored. Enter the position in screen pixels.

RI_WWIDTH

This parameter controls the width of the preview window. Enter the width value in screen pixels.

Understanding the Viewer Status File

While the Viewer is executing, it writes status information into a file (either a table or a text file) and, if the RI_DISPERR flag is **T**, may display error message boxes. For an explanation of the RI_DISPERR flag, see the **Using Control Tables and Files** section.

The status file created by the Viewer is called RRUNOUT.DBF (if you are using a control table) or RRUNOUT.OUT (if you are using a text control file), unless you use the **/O** switch with the Viewer command to specify a different file name. The Viewer creates the status file in the current directory, overwriting any existing status file of the same name.

After calling the Viewer, you should check the status file for information about Viewer processing. If the Viewer encountered an error, the file will contain an error message that explains why a report was canceled, as well as an error code that lets you determine the type of error. The status file also contains the number of reports and pages output, so a report can be restarted where it left off.

To avoid being confused by multiple status files, delete existing status files before calling the Viewer. If you are using the Viewer on a network, use **/O** to specify a unique status file for each user.

Status File Fields

Figure 2.12 lists the fields in the Viewer status file. Each field name has the prefix RO_ to indicate that it provides information about Viewer **O**utput.

Field Name	Contents	Data Type	Width
RO_ECODE	Error code	Character	1
RO_EMSG	Error message	Character	80
RO_PAGES	Number of pages completed	Numeric	9
RO_REPORTS	Number of reports completed	Numeric	5
RO_RIRECNO	Last report processed	Numeric	5

Figure 2.12 Fields in Viewer Status File

RO_ECODE

The error code field contains one of the following characters:

- **N** – the Viewer completed without error; RO_EMSG is blank.
- **C** – the user selected Cancel to cancel a report; the RO_EMSG message is "Report canceled."
- **J** – the Viewer command or the control table contains a syntax error; see the message in the RO_EMSG field and/or the number in the RO_RIRECNO field.
- **R** – the report definition or the value in a control table field caused an error in the report; see the RO_EMSG field.

RO_EMSG

The error message field is blank if Viewer processing completed without an error. If processing was canceled for any reason, this field contains the error message. If RO_ECODE contains **C**, the message is "Report canceled," meaning the user canceled a report. If RO_ECODE contains **R**, the message is the same as the one that displays when you attempt to output the report from within Report Writer.

If RO_ECODE contains **J**, there is an error in the Viewer command or in the control table. If the error is in the control table, the record number of the erroneous record is returned in the RO_RIRECNO field. Use this record number along with the error message to find and correct the error.

RO_REPORTS

This field contains the number of reports that completed successfully. For example, if you call the Viewer to print three reports and the printer jams during the second report, this field contains the number 1, indicating that one report completed successfully. Use this number to determine which report did not complete and to restart it.

RO_PAGES

This entry contains the number of the last page completed in the report (or in the most recently processed report, if the Viewer command specifies multiple reports). If a report terminated due to an error, the entry contains the number of the last page completed before the error occurred. Use this number to restart a canceled report at the page where the error occurred.

For example, if you are printing pages 10 through 20 of a report and the printer jams on page 15, this entry will contain 14 (the number of the last page that printed successfully). If RO_PAGES contains 14, you can restart the report at page 15 by entering 15 in RI_BEGPAGE and 20 in RI_ENDPAGE.

Note that the Viewer does not update the RO_PAGES field after each page unless the value of RI_CHKTIME in the control table is **P**. See the section in this chapter entitled **Creating and Using Control Tables and Files** for a description of RI_CHKTIME.

RO_RIRECNO

This field contains the control table record number of the last report processed, whether or not it completed successfully. You can use this number together with the error message in RO_EMSG to find and correct an error in the control table. This number is also useful for restarting a report that was part of a multi-report job, since it is the control table record number of the last report that was processed.

Application Calls to the Viewer Executable

This section provides examples illustrating how you can incorporate calls to the Viewer in C, Visual Basic, FoxPro for Windows, and PowerBuilder.

Note that these examples are provided only to demonstrate the syntax for calls to the Viewer for each language.

If your application includes logic to check the Viewer status file, keep in mind that up-to-date information from that file will not be available until the Viewer reports complete. To avoid confusion between old and new status files, delete any existing status files before calling the Viewer.

Calling the Viewer from C

The Windows API provides a function named WinExec for executing programs. To call the Viewer from a Windows C program, you could include a function such as the one illustrated in Figure 2.13.

```
BOOL RunThisReport (LPSTR lpRunin, int iReport, LPSTR lpRptlib)
{
    // Run a single report.
    // Input:
    //   lpRunin              pointer to Viewer control table name
    //   iReport              control table record number of report
    //   lpRptlib             pointer to default library directory name
    // Output: FALSE if Windows could not execute the Viewer program

    char szBuffer [128];
    UINT error;

    wsprintf(szBuffer,"rrwrun %s %d /R%s",lpRunin,iReport,lpRptlib);
    if ((error=WinExec((LPSTR)szBuffer, SW_SHOW)) < 32)
    {
        LoadString(hAppInst, EXE_ERR+error, szBuffer, sizeof(szBuffer));
        MessageBox(hAppWnd, szBuffer, szAppname, MB_ICONSTOP);
        return FALSE;
    }
    return TRUE;
}
```

Figure 2.13 Calling the Viewer from a C Program

In this example, the call to the Viewer includes the /R switch to specify a default report library directory. You can include any combination of the command switches explained in the section of this chapter entitled **Command Switches**.

Note that the second parameter supplied to WinExec, SW_SHOW, is ignored if you have included control table values to govern the Viewer display.

Calling the Viewer from Visual Basic

Visual Basic provides a function named Shell that takes two arguments: a command-line string and a Windows display style. The Viewer ignores the second argument if display characteristics are specified by Viewer control table values.

Figure 2.14 illustrates a subroutine that could be used to call the Viewer from a Visual Basic application.

```
Sub Command1_Click ()
  cmd$ = "c:\rrw\rrwrun.exe c:\rrw\rrsample\rrwrunin 1 /WN"
  i% = Shell(cmd$, 1)
End Sub
```

Figure 2.14 Sample Visual Basic Subroutine Calling the Viewer

In this example, the Viewer is executed using the Viewer control table in C:\RRW\RRSAMPLE. The /WN switch specifies that database users will *not* have write access to the database files used by the Viewer reports while those reports are being generated. Note that you can use any combination of Viewer command switches.

Calling the Viewer from Windows FoxPro

You can execute the Viewer from FoxPro for Windows using the FoxPro RUN command. For example, you could include the following line to call the Viewer from a FoxPro for Windows application:

```
RUN /N RRWRUN ACCOUNTS 1 /RC:\DEPT99
```

In this example, the Viewer is executed using a control table named "accounts." The /R switch specifies a default library directory of c:\dept99. The /N switch is used in FoxPro to execute another Windows-based application.

Calling the Viewer from PowerBuilder

You can execute the Viewer from a PowerBuilder script using the SetProfileString command to change any of the parameters in the control file and the RUN command to execute the Viewer. Figure 2.15 illustrates a script that could be used to call the Viewer from a PowerBuilder application.

```
SetProfileString("D:\RRWRUN.IN","rrwrun","ri_copies","2")
run("RRWRUN.EXE /Td:\rrwrun.in /Od:\rrwrun.out")
```

Figure 2.15 Sample PowerBuilder Script Calling the Viewer

Parameter Passing

You can control some features of the layout and content of reports at runtime by prompting users to enter values for parameters, then passing the values to reports. Typically, you prompt the user for a text string or other data item that is not stored in the database. For example, you might prompt the user for his or her name and use the name in a "Report Author" field in the page footer or title.

You can also use parameter passing to control report processing at runtime. You can pass parameters to a report in two ways:

- Define special parameters in the Viewer control table or file. At runtime, prompt the user to enter values that are stored in these parameters. Pass the values to your report using a calculated field whose expression includes the RIPARAM( ) function.
- Create your own menus and prompts and store user entries and selections in a special parameter table. When you create the report, link the parameter table to the report's master table and use values from the parameter table in the report.

Passing Control Parameter Values

Follow these general steps to pass values to reports using control table or control file parameters.

1. In the Viewer control table or file, define parameters for values you want to pass to the report.

2. Prompt the user to enter a value for the parameter in one of two ways:
   - ❑ Create your own menus or prompts within your application.
   - ❑ Enter a question mark as the value of the control table parameter.
3. Incorporate the user's entry into the report using the RIPARAM( ) function in a calculated field expression.

The following sections describe each step in detail.

Defining Control Parameters

In addition to the predefined parameters listed in Figures 2.3 and 2.11, your control table or file can include parameters you define. A user-defined parameter can have any name and can be up to 512 characters wide. You can define as many parameters as you need for your application. The control table or file need not include any user-defined parameters. If any are present, the table or file need not contain values for all of them.

Prompting for User Input

You can get user input in two ways:

- ❑ Supply a menu or prompt in your application that leads the user to supply a value. Store this value in the user-defined parameter in the Viewer control table or file.
- ❑ Enter a question mark (**?**) value for any user-defined parameter. Whenever a user-defined parameter contains a question mark, the user will be prompted to enter a value.

Using the Question Mark Parameter Value

The simplest way to get user input for reports is to use a question mark (**?**) as the value for a user-defined parameter. Optionally, the value can also include the text you want to appear as a prompt.

For example, if you want to prompt the user for his or her name, you might create an AUTHOR parameter in your control table or file and give it the value "?Enter your name:". At runtime, the user will see the dialog box shown in Figure 2.16.

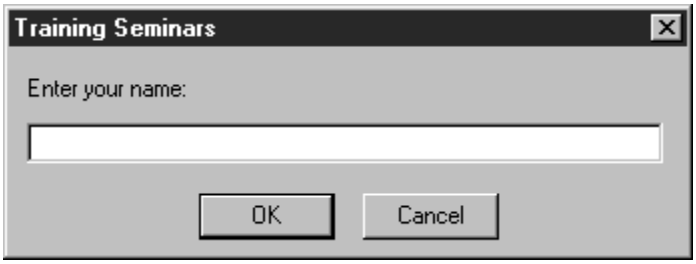

Figure 2.16 Viewer Dialog Box with Prompt

The size and shape of this dialog box is the same for all user-defined parameters. The title bar contains the value of RI_WTITLE. If RI_WTITLE is not specified, the Viewer uses the report name. For control tables, the length of the field in the control table determines the maximum length of the user's entry. For example, if the length in the control table is 30 characters, the Viewer will not allow entry of more than 30 characters. If the user selects Cancel, the report will not run and the Viewer will write the "Canceled" message to the status file.

If your control parameter contains a question mark only and no text string, the Viewer displays the dialog box shown in Figure 2.16 with the prompt "Enter value for (FIELD NAME)."

Incorporating User-Supplied Values in Reports

Once the user has entered or selected a value that is stored in the Viewer control table or file, you pass the value to your report using the RIPARAM( ) function in a calculated field expression. RIPARAM( ) takes a control parameter name as its argument and returns the parameter's value as a string.

For example, in a general ledger application, you might create a control parameter CONAME for the company name, then prompt the user to enter a company name. To use the company name on the report, create a calculated field in Report Writer whose expression is:

```
RIPARAM("CONAME")
```

You can place the calculated field wherever you want the company name to appear on the report.

Although this example uses an RIPARAM( ) calculated field to provide user input as text in the report, you can use such fields to perform many different functions in a report. For example, you might prompt the user for a value for a DISCOUNT field. In the calculated

field on the report, you can convert the user-entered character data to numeric using a calculated field expression such as:

```
ORDERTOT * VAL(RIPARAM("DISCOUNT"))
```

Using a Parameter Table

You can also pass parameters to a report by storing user-supplied values in a separate table called a *parameter table* and relating this table to the report's master table.

To pass parameters to a report using a parameter table, follow these steps:

1. Decide on the parameters you need and create a parameter table using your Xbase software.
2. In interactive Report Writer, create a new report or retrieve and modify an existing report.
3. Create a calculated field that you will use to link the report's master table to the parameter table.
4. Relate the master table to the parameter table using the calculated field as the linking field from the master table.
5. Use the fields from the parameter table in your report and save the report.
6. Create an application program that will get the information from the user, store it in the parameter table, and call the Viewer to generate the report.

The following sections describe steps 1 through 4 in more detail and provide an example of using a parameter table in an application.

Creating the Parameter Table

A parameter table can contain as many parameters as you need for your application. For example, you may want the user to supply a date, a range of dates, an account number, a category name, a list of items, or a logical true/false flag. Include one field for each parameter. You can use any data type. Assign a field width sufficient for your purpose.

Most likely, your parameter table will have a single record. However, if you want to create one parameter table for use with multiple reports,

the table should have a record for each report. For each report, append one blank record to contain the parameters.

Creating the Calculated Linking Field

In interactive Report Writer, use Calculations ⇒ Calculated Field to create a calculated linking field whose expression is a constant equal to the record number or index key value for the appropriate record in the parameter table. If your parameter table has a single record, you can simply use the number 1 as the calculated linking field value.

If your parameter table has multiple records, you might use the key field value as the calculated linking field value. For example, if the parameter table's key field is the report name, the calculated field's expression is the report name that identifies the appropriate record, as in "INVOICE1."

Relating the Master Table and the Parameter Table

Use Database ⇒ Relations to relate the master table and the parameter table. Use the calculated field as the linking field from the master table and the record number or key field as the linking field from the parameter table.

Chapter 3
Accessing the Viewer DLL

Introduction

This chapter explains how to use the Viewer DLL to run reports from within Windows application programs. As noted in Chapter 1, the Viewer DLL provides one of three methods for running reports using the Viewer. The other methods are explained in Chapter 2, "Using the Viewer Executable," and Chapter 4, "Using the Custom Control."

The Viewer DLL provides a direct application programming interface to the Viewer. The general logic of using this API to invoke the Viewer is to cycle through the following four steps as many times as you wish:

- Select a report or report/library combination with **chooseReport** or **getRuntimeRecord**. Or, select a report library with **getNewReportHandle** and **setLibrary**.
- Use various routines to get and set runtime control parameters.
- Use **writeRuntimeRecord** to save the parameters in a Viewer Control File for later execution or **execRuntime** to use the Viewer to run the report immediately.
- Clean up the current report with **endReport**.

The routines provided by this API are grouped into five categories:

- Action Routines
- Get-Parameter Routines
- Set-Parameter Routines
- User-Interface Routines
- Error-Handling Routines

Action Routines

Action routines are used to begin working with the runtime DLL or a specific report, to run a report, and to free resources used in working with a report or the runtime DLL as a whole.

- **chooseReport** specifies a report or library/report combination.

- **endReport** cleans up resources associated with a given report.
- **execRuntime** runs a given report.
- **getNewReportHandle** obtains the handle of an empty report-information structure.
- **getRuntimeRecord** specifies a library/report combination along with parameter values as defined in a Viewer Control File record.
- **writeRuntimeRecord** writes to a Viewer Control File record the current parameter values associated with a given report.

Get-Parameter Routines

Get-parameter routines are used to obtain the values of various parameters as they were saved with the report, or as they have been overridden by values from a Viewer Control File or by previous uses of set-parameter routines. It is important to understand the concept of the "current" value of a parameter.

- If you have initiated the processing of a report via a call to **chooseReport** and have not yet used a set-parameter routine for a given parameter, the current value of that parameter is the value saved in the report. Once you have used a set-parameter routine for the parameter, the current value is the value you specified via the set-parameter routine.
- If you have initiated the processing of a report via a call to **getRuntimeRecord** and have not yet used a set-parameter routine for a given parameter, the current value of the parameter is the value saved in the report unless the parameter is overridden in the runtime control file record, in which case the current value is the value from the control file record. Once you have used a set-parameter routine for the parameter, the current value is the value you specified via the set-parameter routine.
- If you have initiated the processing of a report via a call to **getNewReportHandle** and have not yet used a set-parameter routine for a given parameter, the current value of the parameter is the default value of that parameter. Once you have used a set-parameter routine for the parameter, the current value is the value you specified via the set-parameter routine.

All get-parameter routines return the current values of parameters. Once you have used the set-parameter routine for a given parameter,

there is no way to get a previous value. If you need to be able to get original values, use **chooseReport** and then use get-parameter routines to get the original values. Your program must remember the original values once it begins using set-parameter routines to override them. Alternatively, use **chooseReport** and remember your overrides instead of calling set-parameter routines. Then call the set-parameter routines just before calling **execRuntime** or **writeRuntimeRecord**.

- **getBeginPage** gets the value of the starting-page-number parameter.
- **getCopies** gets the value of the number-of-copies parameter.
- **getDisplayErrors** gets the value of the display-errors flag.
- **getDisplayStatus** gets the value of the display-status-window flag.
- **getEndPage** gets the value of the ending-page-number parameter.
- **getExportDest** gets the value of the export-destination flag.
- **getFilter** gets the filter expression.
- **getFilterUsage** gets the value of the filter-usage flag.
- **getFirstFieldName** gets the name of the first field from tables used in the report.
- **getFirstFilteredFieldName** gets the name of the first field suitable for use as a sort or group field.
- **getFirstGroupField** gets the name of the first group field of the report.
- **getFirstRelationInfo** gets the values of parameters pertaining to the first related table used in the report.
- **getFirstSortField** gets the name of the first sort field of the report.
- **getFirstUserParam** gets the name of the first user-parameter used in the report.
- **getHighScope** gets the value of the high-scope parameter.
- **getLibrary** gets the name of the report library parameter.
- **getLowScope** gets the value of the low-scope parameter.
- **getMasterIndexInfo** gets the value of the master-index parameter.
- **getMasterTableName** gets the name of the master table used in the report.

- **getMemoName** gets the name of the ASCII memo file used in the report.
- **getNextFieldName** gets the name of the next field from tables used in the report.
- **getNextFilteredFieldName** gets the name of the next field suitable for use as a sort or group field.
- **getNextGroupField** gets the name of the next group field of the report.
- **getNextRelationInfo** gets the values of parameters pertaining to the next relation used in the report.
- **getNextSortField** gets the name of the next sort field of the report.
- **getNextUserParam** gets the name of the next user-parameter used in the report.
- **getOutputDest** gets the value of the output-destination parameter.
- **getOutputFile** gets the name of the output file.
- **getPreventEscape** gets the value of the prevent-user-escape flag.
- **getPrinter** gets the name of the current printer.
- **getPrinterPort** gets the name of the current printer port.
- **getReportPick** gets the value of the report-selection flag.
- **getScopeUsage** gets the value of the scope-usage flag.
- **getStatusEveryPage** gets the value of the report-status-frequency flag.
- **getTestPattern** gets the value of the print-test-pattern flag.
- **getWinTitle** gets the value of the window-title parameter.

Set-Parameter Routines

Set-Parameter routines are used to override the existing values of various report parameters. Once you have called a given set-parameter routine, the value returned by the corresponding get-parameter routine will be the value most recently set for that parameter.

- **setBeginPage** sets the value of the starting-page-number parameter.
- **setCopies** sets the value of the number-of-copies parameter.
- **setDataDir** specifies an override for the default data directory.

- **setDisplayErrors** specifies whether to display errors.
- **setDisplayStatus** specifies whether to display a status window.
- **setEndPage** sets the value of the ending-page-number parameter.
- **setExportDest** sets the value of the export-destination flag.
- **setFilter** specifies a filter expression.
- **setFilterUsage** sets the value of the filter-usage flag.
- **setGroupField** sets the name of a group field.
- **setHighScope** sets the value of the high-scope parameter.
- **setImageDir** specifies an override for the default image directory.
- **setIndexExtension** specifies a default index filename extension.
- **setLibrary** specifies a report-library.
- **setLibraryDir** specifies an override for the default report/library directory.
- **setLowScope** sets the value of the low-scope parameter.
- **setMasterIndexInfo** specifies master-index parameters.
- **setMasterTableName** sets the name of the master table used in the report.
- **setMemoName** sets the name of the ASCII memo file used in the report.
- **setOutputDest** sets the output-destination flag.
- **setOutputFile** sets the name of the output file.
- **setPreventEscape** specifies whether the user should be allowed to terminate the report.
- **setPrinter** specifies the name of the printer to be used in generating a report.
- **setPrinterPort** specifies the name of the port to be used for printing.
- **setRelationInfo** sets the values of parameters pertaining to a related table used in the report.
- **setReportPick** specifies the optional use of a report-selection dialog in the Viewer that allows the user to select one or more reports at runtime.
- **setScopeUsage** specifies the value of the scope-usage flag.
- **setSortField** specifies the name of a sort field.

- **setStatusEveryPage** specifies how often report status should be returned.
- **setStatusFileName** specifies the filename for returning status information from the Viewer executable.
- **setSuppressTitle** specifies whether to print Title and Summary areas of reports when no records are found.
- **setTestPattern** specifies whether to generate a test pattern.
- **setUserParam** specifies a value for a user-parameter used in the report.
- **setWinBorderStyle** sets the style of the preview window border.
- **setWinControlBox** specifies whether the preview window should include a control box.
- **setWinHeight** specifies the height of the preview window.
- **setWinLeft** specifies the position of the left edge of the preview window.
- **setWinMaxButton** specifies whether the preview window should include a maximize button.
- **setWinMinButton** specifies whether the preview window should include a minimize button.
- **setWinTitle** specifies the window title to be used in certain Viewer windows.
- **setWinTop** specifies the position of the top edge of the preview window.
- **setWinWidth** specifies the width of the preview window.
- **setWriteAllow** specifies whether to allow write-access to files being used by the Viewer executable.
- **setXbaseEditor** specifies whether memo files were created with an Xbase memo editor.

User-Interface Routines

User-Interface routines use Windows dialogs to present the user with a list of alternatives for various report parameters.

- **choosePrinter** is used to present the user with a dialog from which to select a printer.

Error-Handling Routines

The Error-Handling routines are used to obtain information about errors resulting from calls to the other routines.

- **getErrorInfo** is used to obtain an error code and/or error text relating to the most recent error condition.
- **resetErrorInfo** is used to reset the current value of the error code and error text. This is useful if you only check for errors after certain calls and want to be certain that the error status you obtain via **getErrorInfo** is not from some previous call.

Important Note

Many functions provided by this API require the calling routine to specify one or more addresses at which to return values. For each such pointer argument, the calling routine also specifies the size of the buffer pointed to. It is essential that the indicated size be no larger than the actual size of the buffer pointed to. The Viewer DLL usually returns precisely the number of bytes specified by the buffer size, if necessary padding with null bytes to fill the buffer. If the calling routine specifies a size that is larger than its actual buffer, unpredictable bad things will happen.

Functions Provided by the Viewer DLL

The following sections present detailed descriptions of the functions provided by the Viewer DLL API. The functions are listed in alphabetical order. For a listing of functions by category, see the preceding section of this chapter. Each function description begins with a function prototype, which is followed by a brief description of each argument, a list of values returned by the function, a function description, a list of related functions, and an example in C of a call to the function.

The API for the Viewer DLL is defined in two header files, one named RRRPT32.H, for use in C/C++ programs, and one named RRDECL32.BAS for use in Visual Basic programs.

choosePrinter

BOOL FAR PASCAL choosePrinter(int *hReport*, **LPSTR** *lpszPrinter*, **int** *prSize*, **LPSTR** *lpszPort*, *int poSize*);

hReport	Report handle.
lpszPrinter	Address of buffer in which to return selected printer name.
prSize	Size of buffer pointed to by *lpszPrinter*.
lpszPort	Address of buffer in which to return selected printer port.
poSize	Size of buffer pointed to by *lpszPort*.

Return Value

The **choosePrinter** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **choosePrinter** to allow the user to interactively select a new printer and printer port. The name of the printer selected by the user will be returned in the buffer specified by *lpszPrinter* to the extent allowed by *prSize*. The name of the printer port selected by the user will be returned in the buffer specified by *lpszPort* to the extent allowed by *poSize*.

Related Functions

setPrinter, setPrinterPort, getPrinter, getPrinterPort

Example

To allow the user to select a new printer and printer port for the report whose handle is *hRpt* and then apply those selections to the report:

```
{
   char prbuf[100];
   char pobuf[10];
   choosePrinter (hRpt, (LPSTR)prbuf, 100, (LPSTR)pobuf, 10);
   setPrinter (hRpt, (LPSTR)prbuf);
   setPrinterPort (hRpt, (LPSTR)pobuf);
}
```

chooseReport

int FAR PASCAL chooseReport (LPSTR *lpszAppName*, **LPSTR** *lpszLibName*, **int** *lSize* **LPSTR** *lpszRepName*, **int** *rSize*);

lpszAppName	Name of calling application.
lpszLibName	Name of report library, or buffer in which to return name of library, if any.
lSize	Size of *lpszLibName* buffer.
lpszRepName	Name of report, or buffer in which to return name of report.
rSize	Size of *lpszRepName* buffer.

Return Value

The **chooseReport** function returns a report-information handle if there are no errors. A return value of zero indicates an error. To obtain more information about the error use **getErrorInfo** with a handle of zero.

Description

The *lpszLibName* argument specifies the name of a report library, points to an empty buffer, or is a NULL pointer. The *lpszRepName* argument specifies the name of a report file, specifies the name of a report contained in a report library, points to an empty buffer, or is a NULL pointer. How **chooseReport** interprets these arguments is described below.

If *lpszLibName* and *lpszRepName* both point to non-empty buffers, **chooseReport** opens the library specified by *lpszLibName*, reads the report specified by *lpszRepName*, and prepares a report-information structure based on that report.

If *lpszLibName* points to a non-empty buffer and *lpszRepName* points to an empty buffer or is NULL, **chooseReport** assumes *lpszLibName* contains the name of a report library and presents a dialog via which the user can select a report from those available in the specified library. After the user selects a report, **chooseReport** opens the specified library, reads the selected report, copies the selected report name into *lpszRepName* to the extent allowed by *rSize* (unless *lpszRepName* is NULL or *rSize* is zero), and prepares a report-information structure based on that report.

If *lpszRepName* points to a non-empty buffer and *lpszLibName* points to an empty buffer, **chooseReport** assumes *lpszRepName* contains the name of a report file, reads the report from the specified file, and prepares a report-information structure based on that report.

If both *lpszLibName* and *lpszRepName* point to empty buffers or are NULL, **chooseReport** displays a File-Open dialog, via which the user can select a report file or a report library. If the user selects a report file, **chooseReport** reads the selected report, copies the report name into *lpszRepName* to the extent allowed by *rSize* (unless *lpszRepName* is NULL or *rSize* is zero), and prepares a report-information structure based on that report. If the user selects a report library, **chooseReport** presents a dialog via which the user can select a report from the list of reports available in the selected library. After the user selects a report, **chooseReport** opens the library, reads the selected report, copies the library name into *lpszLibName* to the extent allowed by *lSize* (unless *lpszLibName* is NULL or *lSize* is zero) and the report name into *lpszRepName* to the extent allowed by r*Size* (unless *lpszRepName* is NULL or *rSize* is zero), and prepares a report-information structure based on that report.

If *lpszLibName* or *lpszRepName* (when interpreted as a report file name) does not include a path, the Viewer looks for the a file of that name in the default library directory specified in RRW.INI. If no default is specified in the INI file either, the Viewer looks for the file in the current directory.

The handle returned by **chooseReport** is used as input to most other functions contained within this API. The *lpszAppName* argument identifies the calling application.

Related Functions

endReport, getRuntimeRecord

Example

To allow the user to select a new report from the library c:\libs\reports.rp5 :

```
{
   char RepName[40];
   char szLibName[]="c:\\libs\\reports.rp5";
   int hRpt;

   hRpt = chooseReport (NULL,szLibName,sizeof(szLibName),
   szRepName,sizeof(szRepName));
}
```

endReport

BOOL FAR PASCAL endReport (int *hReport***);**

hReport	Report handle.

Return Value

The **endReport** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Call the **endReport** function to signify that your application is finished with the report associated with *hReport*. This enables **endReport** to clean up resources associated with that report.

Related Functions

chooseReport, getRuntimeRecord, **getNewReportHandle**

Example

To inform the DLL that you are finished with the report whose handle is *hRpt*:

```
endReport (hRpt);
```

execRuntime

BOOL FAR PASCAL execRuntime (int *hReport***, BOOL** *bWait***, int** *cmdShow***, LPINT** *lpiECode***, LPLONG** *lplPageCount***, LPSTR** *lpszEMsg***, int** *emSize***);**

hReport	Report handle.
bWait	Synchronous operation flag.
cmdShow	Windows **ShowWindow** value.
lpiECode	Error-code buffer.
lplPageCount	Page-count buffer.
lpszEMsg	Error-message buffer.
emSize	Size of *lpszEMsg* buffer.

Return Value

The **execRuntime** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

After using **chooseReport**, **getRuntimeRecord**, or **getNewReportHandle** to prepare a report-information structure and using other functions provided by this API to modify the structure's contents, use **execRuntime** to run the report. If *bWait* is zero, **execRuntime** will invoke RRWRUN.EXE to begin execution of the report and then return. If *bWait* is non-zero, **execRuntime** will not return until the report execution is complete, in which case the buffers provided by *lpiECode*, *lpiPageCount*, and *lpszEMsg* will be used to return status.

If *bWait* is non-zero, *lpiECode* will contain one of the following characters when **execRuntime** returns:

N	Successful execution of the requested report.
C	The user canceled the report. *lpszEMsg* will contain "Report canceled."
J	The report structure identified by *hReport* contains inconsistent or incorrect information. *lpszEMsg* will contain an error message describing the problem.
R	The requested report began to execute, but failed to complete successfully. *lpszEMsg* will contain an error message describing the problem.

Regardless of the value of *bWait*, any error resulting from the failure of **execRuntime** to invoke the Viewer executable is available through **getErrorInfo**.

If you have used **setStatusEveryPage** to request that the runtime status be updated after every page and *bWait* is non-zero, *lpiPageCount* will contain the number of the last page completed in the report. If the report did not complete successfully, *lpiPageCount* contains the number of the last page completed before the error occurred. Use this number to restart an incomplete report at the page where the error occurred. For example, if *lpiPageCount* is 14, you can use **setBeginPage** to restart the same report at page 15. (Use **setEndPage** to set the ending page to 999999999.)

If *bWait* is zero, **execRuntime** leaves *lpiECode*, *lpiPageCount*, and *lpiEMsg* unchanged. In this case, the Viewer will create a runtime status file and the information provided by *lpiECode*, *lpiPageCount*, and *lpiEMsg* is instead provided by the fields RO_ECODE, RO_PAGES, and RO_EMSG. See Chapter 2 for details of the runtime status file.

See Windows SDK documentation for the ShowWindow( ) function for information about legal values of *cmdShow*.

Related Functions

chooseReport, getRuntimeRecord, getNewReportHandle, setBeginPage, setEndPage, setStatusEveryPage

Example

To synchronously run the report whose handle is *hRpt* and test the results:

```
{
   int ecode;
   long pages;
   char emsg[200];
   int done = FALSE;
   while (!done)
   {
      // code to let user make changes to parameters, etc.
      execRuntime (hRpt,      // report handle
                   1,         // synchronous
                   SW_SHOW, // current size/position
                   (LPINT)&ecode,    // place for error code
                   (LPLONG)&pages,   // ... pages printed
                   (LPSTR)emsg,      // ... error message
                   200);     // size of emsg buffer
      switch (ecode)
      {
         case 'N':      // success
         case 'C':      // user canceled report
            done = 1;   // either way, we're happy
            break;
         case 'J':      // problem with parameters
            // error handling code
            break;
         case 'R':      // problem running report
            // error handling code
            break;
      } // end switch
   } // end while
}
```

getBeginPage

BOOL FAR PASCAL getBeginPage (int *hReport*, **LPLONG** *lplBeginPage*);

hReport	Report handle.
lplBeginPage	Starting-page-number buffer.

Return Value

The **getBeginPage** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getBeginPage** to obtain the current value of the "starting page" parameter. **getBeginPage** returns the current value of the starting page number in the form of a long integer in the buffer pointed to by *lplBeginPage*. See **setBeginPage** for a discussion of this parameter.

Related Functions

setBeginPage, getEndPage, setEndPage, execRuntime

Example

To get the current starting page for the report whose handle is *hRpt*:

```
{
   LONG begPage;

   getBeginPage (hRpt, (LPLONG)&begPage);
}
```

getCopies

BOOL FAR PASCAL getCopies (int *hReport*, **LPINT** *lpiCopies*);

hReport	Report handle.
lpiCopies	Number-of-copies buffer.

Return Value

The **getCopies** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getCopies** to obtain the current value of the "number of copies" parameter for the report specified by *hReport*. **getCopies** returns the number of copies in the form of an integer in the buffer pointed to by *lpiCopies*. See **setCopies** for a discussion of this parameter.

Related Functions

setCopies

Example

To get the current number of copies for the report whose handle is *hRpt*:

```
{
   int copies;
   getCopies (hRpt, (LPINT)&copies);
}
```

getDisplayErrors

BOOL FAR PASCAL getDisplayErrors (int *hReport*, **BOOL FAR *** *lpbDispErr*);

hReport	Report handle.
lpbDispErr	Display-errors-flag buffer.

Return Value

The **getDisplayErrors** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getDisplayErrors** to obtain the current value of the "display errors" parameter for the report specified by *hReport*. **getDisplayErrors** returns the parameter in the form of a

boolean in the buffer pointed to by *lpbDispErr*. See **setDisplayErrors** for a discussion of this parameter.

Related Functions

setDisplayErrors

Example

To get the display-errors flag for the report whose handle is *hRpt*:

```
{
   BOOL bDispErrors;
   getDisplayErrors (hRpt, (BOOL FAR *)&bDispErrors);
}
```

getDisplayStatus

BOOL FAR PASCAL getDisplayStatus (int *hReport*, **BOOL FAR *** *lpbDispStatus*);

hReport	Report handle.
lpbDispstatus	Display-status-flag buffer.

Return Value

The **getDisplayStatus** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getDisplayStatus** to obtain the current value of the "display status" parameter for the report specified by *hReport*. **getDisplayStatus** returns the parameter in the form of a boolean in the buffer pointed to by *lpbDispStatus*. See **setDisplayStatus** for a discussion of this parameter.

Related Functions

setDisplayStatus, getPreventEscape, setPreventEscape

Example

To get the display-status flag for the report whose handle is *hRpt*:

```
{
   BOOL dispStatus;
   getDisplayStatus (hRpt, (BOOL FAR *)&dispStatus);
}
```

getEndPage

BOOL FAR PASCAL getEndPage (int *hReport*, **LPLONG** *lplEndPage*);

hReport	Report handle.
lplEndPage	Ending-page-number buffer.

Return Value

The **getEndPage** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getEndPage** to obtain the current value of the "ending page" parameter for the report specified by *hReport*. **getEndPage** returns the current value of the ending page number in the form of a long integer in the buffer pointed to by *lplEndPage*. See **setEndPage** for a discussion of this parameter.

Related Functions

setEndPage, getBeginPage, setBeginPage

Example

To get the current ending page for the report whose handle is *hRpt*:

```
{
   LONG endPage;
   getEndPage (hRpt, (LPLONG)&endPage);
}
```

getErrorInfo

BOOL FAR PASCAL getErrorInfo (int *hReport,* **LPSTR** *lpszMsg***, int** *mSize***, LPINT** *lpiCode***);**

hReport	Report handle.
lpszMsg	Error-text buffer.
mSize	Size of *lpszMsg* buffer.
lpiCode	Error-code buffer.

Return Value

The **getErrorInfo** function returns a non-zero value if it is returning error information in *lpszMsg* and/or *lpiCode*. It returns zero if no error has occurred about which it can return information.

Description

Use **getErrorInfo** to obtain information about the most recent error condition relating to the report indicated by *hReport*. (If the error is a result of a call to **chooseReport**, **getRuntimeRecord**, or **getNewReportHandle**, you will not have a valid report handle and should use a handle of zero.) When other routines in this API indicate that an error has occurred by returning a zero value, you can use **getErrorInfo** to get details. **getErrorInfo** returns an error message in the buffer pointed to by *lpszMsg* to the extent allowed by *mSize*, unless *lpszMsg* is NULL or *mSize* is negative or zero. **getErrorInfo** returns an error code in the buffer pointed to by *lpiCode* unless *lpiCode* is NULL.

getErrorInfo returns one of the following values in *lpiCode*:

- **C** (Cancel) indicates that the user canceled out of a dialog presented by the Viewer DLL.
- **D** (Diagnostic) indicates a miscellaneous error such as insufficient memory.
- **I** (Iteration) indicates that there are no more values for the requested **getFirst...** or **getNext...** function. This is not really an error condition. It would be returned after the third and subsequent calls to **getNextSortField** in a report containing two sort fields, for example.
- **J** (Job Control) indicates a problem with the Viewer Control File specified as *lpszControlFile* to **getRuntimeRecord**.
- **L** (Library) indicates a problem with a report library being processed by the Viewer DLL. It would be returned, for example, if **chooseReport** were unable to read the report library specified as *lpszLibName*.
- **S** (Syntax) indicates a problem with the arguments passed to the routine generating the error. This might indicate NULL passed for a required pointer or a buffer size of zero, for example.
- **V** (Value) indicates that no value is available for the parameter whose value you have requested.

The information returned by **getErrorInfo** will pertain to the most recent error resulting from a call to the runtime DLL involving the specified report handle. The DLL does not clear its internal error status on entry to its routines. For this reason, you should test for errors after each call, chain calls together in a single *if* statement with an error handler for the compound statement, or use **resetErrorInfo** before any calls for which you are interested in obtaining error status. Since **resetErrorInfo** always returns non-zero, you can safely begin a chain of calls with a call to **resetErrorInfo**, as in the example below.

Related Functions

resetErrorInfo

Example

```
if (resetErrorInfo(...)    // reset error status
 && setLibrary (...)
 && setMasterTable (...)
 && setFilter (...)
 && setFilterUsage (...)
   )
   execRuntime (hRpt, ...);   // sets went ok; run report
else     // error on one of the sets, check it out
{
   char emsg[200];
   int ecode;
   getErrorInfo (hRpt, (LPSTR)emsg, 200, (LPINT)&ecode);
   // ecode will have an error code
   // emsg will have an error message, truncated to 200
   //  bytes, if necessary
   // code to do something with this error info
}
```

getExportDest

BOOL FAR PASCAL getExportDest (int *hReport*, **LPSTR** *lpszVal*);

hReport	Report handle.
lpszVal	Export-destination-flag buffer.

Return Value

The **getExportDest** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getExportDest** to obtain the current value of the "export destination" parameter for the report specified by *hReport*. See **setExportDest** for a discussion of this parameter.

Related Functions

setExportDest

Example

To get the current export destination for the report whose handle is *hRpt*:

```
{
   char dest[2];
   getExportDest (hRpt, (LPSTR)dest);
}
```

getFilter

BOOL FAR PASCAL getFilter (int *hReport*, **LPSTR** *lpszFilter*, **int** *fSize*);

hReport	Report handle.
lpszFilter	Filter buffer.
fSize	Size of *lpszFilter* buffer.

Return Value

The **getFilter** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getFilter** to obtain the current value of the "filter" parameter for the report specified by *hReport*. **getFilter** returns the current filter (in the form of a valid calculated field expression) in the buffer pointed to by *lpszFilter*, to the extent allowed by *fSize*. If **setFilter** has not previously been used to override the filter saved with the report, **getFilter** returns a logical expression equivalent to the filter defined via the Query dialog in Report Writer or overridden in the Viewer Control File record if *hReport* was obtained via a call to **getRuntimeRecord**. If **setFilter** has been called to override the filter saved with the report, **getFilter** simply returns the value previously set. See **setFilter** for details

of filter expressions. See **setFilterUsage** for details of the interaction between values set by **setFilterUsage** and **setFilter**.

Related Functions

setFilter, getFilterUsage, setFilterUsage

Example

To get the current filter for the report whose handle is *hRpt*:

```
{
   char filter[2000];
   getFilter (hRpt, (LPSTR)filter, 2000);
}
```

getFilterUsage

BOOL FAR PASCAL getFilterUsage (int *hReport*, **LPSTR** *lpszVal*);

hReport	Report handle.
lpszVal	Filter-usage-flag buffer.

Return Value

The **getFilterUsage** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getFilterUsage** to obtain the current value of the "filter usage" parameter for the report specified by *hReport*. **getFilterUsage** returns the current value in the form of a single character in the buffer pointed to by *lpszVal*. See **setFilterUsage** for a discussion of filter-usage values and the interaction between values set by **setFilterUsage** and **setFilter**.

Related Functions

setFilterUsage, getFilter, setFilter

Example

To get the current filter-usage flag for the report whose handle is *hRpt*:

```
{
   char filterUsage[2];
   getFilterUsage (hRpt, (LPSTR)filterUsage);
}
```

getFirstFieldName

BOOL FAR PASCAL getFirstFieldName (int *hReport*, **LPSTR** *lpszFieldName*, **int** *fnSize*);

hReport	Report handle.
lpszFieldName	Fieldname buffer.
fnSize	Size of *lpszFieldName* buffer.

Return Value

The **getFirstFieldName** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getFirstFieldName** to get the first fieldname available for use in the report specified by *hReport*. **getFirstFieldName** returns the fieldname with alias qualifier in the buffer pointed to by *lpszFieldName* to the extent allowed by *fnSize*. Use **getNextFieldName** in a loop to get the rest of the available fieldnames. See **getErrorInfo** for information about how to detect end-of-list.

Related Functions

getFirstFilteredFieldName, getNextFieldName

Example

To get the fieldnames available for the report whose handle is *hRpt*, and add them to the combo box whose handle is *hCombo*:

```
int InitFieldCombo(HWND hCombo, int hRpt)
{
   char szField[80];    // buffer for field name
   int nFields = 1;     // return count of fields
   // Extract field names from the report.
   if (getFirstFieldName(hRpt, szField, sizeof(szField)))
   {
      ComboBox_AddString(hCombo, szField);
      while (getNextFieldName(hRpt, szField, sizeof(szField)))
      {
         ComboBox_AddString(hCombo, szField);
         // This returns false if not an iterator error.
         if (!getError())
            return FALSE;
         nFields++;
      }
   }
   else return getError(); // Error handling routine.
   return nFields;
}
```

getFirstFilteredFieldName

BOOL FAR PASCAL getFirstFilteredFieldName (**int** *hRepstf*, **LPSTR** *lpszFieldName*, **int** *fnSize*, **int** *filter*);

hReport	Report handle.
lpszFieldName	Fieldname buffer.
fnSize	Size of lpszFieldName buffer.
filter	Filter ID.

Return Value

The **getFirstFilteredFieldName** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getFirstFilteredFieldName** to get the first fieldname available for use in the report specified by *hReport* that is suitable for use in the context specified by *filter*. **getFirstFieldName** returns the filename with alias qualifier in the buffer pointed to by *lpszFieldName* to the extent allowed by *fnSize*. Use **getNextFilteredFieldName** in a loop to get the rest of the available fieldnames suitable for use in the specified context. See **getErrorInfo** for information about how to detect end-of-list.

The *filter* argument specifies the context to be used in deciding which available fields to return. The valid values for *filter*, defined in rreport.h, are FILTER_ID_SORT or FILTER_ID_GROUP which return fields suitable for use as sort or group fields, respectively.

Related Functions

getNextFilteredFieldName, getFirstFieldName, getNextFieldName

Example

See **getFirstFieldName** for an example of adding fieldnames to a combo box. To modify that example to get suitable sort fields, simply change the function names from getFirstFieldName and getNextFieldName to getFirstFilteredFieldName and getNextFilterFieldName and add a new last argument to both of FILTER_ID_SORT.

getFirstGroupField

BOOL FAR PASCAL getFirstGroupField (int *hReport*, **LPSTR** *lpszName*, **int** *nSize*);

hReport	Report handle.
lpszName	Group-field-name buffer.
nSize	Size of *lpszName* buffer.

Return Value

The **getFirstGroupField** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getFirstGroupField** and **getNextGroupField** to obtain the current values of the "group field" parameters in the report specified by *hReport*. **getFirstGroupField** returns the name of the first group field in the buffer pointed to by *lpszName*, to the extent allowed by *nSize*. Use **getNextGroupField** iteratively to get the names of the second through eighth group fields. Whenever **getFirstGroupField** is called, the next call to **getNextGroupField** will return the name of the second group field. See **setGroupField**

for a discussion of the group field parameters. See **getErrorInfo** for information about how to detect end-of-list.

Related Functions

getNextGroupField, **setGroupField**, **getFirstSortField**, **getNextSortField**, **setSortField**

Example

To get the names of the group fields for the report whose handle is *hRpt*:

```
{
   char *g[8];
   char g1[80], g2[80], g3[80], g4[80], g5[80], g6[80], g7[80], g8[80];
   int i;
   g[0] = g1; g[1] = g2; g[2] = g3; g[3] = g4;
   g[4] = g5; g[5] = g6; g[6] = g7; g[7] = g8;
   getFirstGroupField (hRpt, (LPSTR)g1, 80);
   for (i = 1; i < 8; i++)
      getNextGroupField (hRpt, (LPSTR)(g[i]), 80);
}
```

getFirstRelationInfo

BOOL FAR PASCAL getFirstRelationInfo (int *hReport*, **LPSTR** *lpszFilePath*, **int** *fSize*, **LPSTR** *lpszIndexPath*, **int** *iSize*, **LPSTR** *lpszTag*, **int** *tSize*, **LPSTR** *lpszAlias*, **int** *aSize*);

hReport	Report handle.
lpszFilePath	Related-filename buffer.
fSize	Size of *lpszFilePath* buffer.
lpszIndexPath	Index-name buffer.
iSize	Size of *lpszIndexPath* buffer.
lpszTag	Tag-name buffer.
tSize	Size of *lpszTag* buffer.
lpszAlias	Alias buffer.
aSize	Size of *lpszAlias* buffer.

Return Value

The **getFirstRelationInfo** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getFirstRelationInfo** to obtain information about the "first" related table in the report specified by *hReport*. **getFirstRelationInfo** returns the related table's name in the buffer pointed to by *lpszFilePath* to the extent allowed by *fSize*; the index and index tag, if any, used in the relation in the buffers pointed to by *lpszIndexPath* and *lpszTag* to the extent allowed by *iSize* and *tSize*; and the alias of the related table in the buffer pointed to by *lpszAlias* to the extent allowed by *aSize*. Use **getNextRelationInfo** in a loop to obtain

equivalent information about the rest of the related tables. See **getErrorInfo** for information about how to detect end-of-list.

Related Functions

getNextRelationInfo, **setRelationInfo**, **getMasterTableName**, **getMasterIndexInfo**

Example

To get information about the first related table in the report whose handle is *hRpt*:

```
{
   char table[80];
   char index[80];
   char tag[20];
   char alias[10];
   getFirstRelationInfo (hRpt, (LPSTR)table, 80, (LPSTR)index, 80,
      (LPSTR)tag, 20, (LPSTR)alias, 10);
}
```

getFirstSortField

BOOL FAR PASCAL getFirstSortField (int *hReport***, LPSTR** *lpszName***, int** *nSize***);**

hReport	Report handle.
lpszName	Sort-field-name buffer.
nSize	Size of *lpszName* buffer.

Return Value

The **getFirstSortField** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getFirstSortField** and **getNextSortField** to obtain the current values of the "sort field" parameters in the report specified by *hReport*. **getFirstSortField** returns the name and direction of the first sort field in the buffer pointed to by *lpszName*, to the extent allowed by *nSize*. Use **getNextSortField** iteratively to get the names and directions of the second through eighth sort fields. Whenever **getFirstSortField** is called, the next call to **getNextSortField** will return the name of the second sort field. See **setSortField** for a discussion of the sort field parameters and a description of the syntax of *lpszName*. See **getErrorInfo** for information about how to detect end-of-list.

Related Functions

getNextSortField, **setSortField**, **getFirstGroupField**, **getNextGroupField**, **setGroupField**

Example

To get the names of the sort fields for the report whose handle is *hRpt*:

```
{
   char *s[8];
   char s1[80], s2[80], s3[80], s4[80], s5[80], s6[80], s7[80], s8[80];
   int i;
   s[0] = s1; s[1] = s2; s[2] = s3; s[3] = s4;
   s[4] = s5; s[5] = s6; s[6] = s7; s[7] = s8;
   getFirstSortField (hRpt, (LPSTR)s1, 80);
   for (i = 1; i < 8; i++)
      getNextSortField (hRpt, (LPSTR)(s[i]), 80);
}
```

getFirstUserParam

BOOL FAR PASCAL getFirstUserParam (int *hReport*, **LPSTR** *lpszName*, **int** *nSize*, **LPSTR** *lpszValue*, **int** *vSize*);

hReport	Report handle.
lpszName	Parameter-name buffer.
nSize	Size of *lpszName* buffer.
lpszValue	Parameter-value buffer.
vSize	Size of *lpszValue* buffer.

Return Value

The **getFirstUserParam** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getFirstUserParam** to get the name and current value, if any, of the first user parameter for the report specified by *hReport*. The name of the user parameter is returned in the buffer pointed to by *lpszName* to the extent allowed by *nSize*. The current value, if any, is returned in the buffer pointed to by *lpszValue* to the extent allowed by *vSize*. Use **getNextUserParam** in a loop to get the names and values of the other user parameters. Use **setUserParam** to give a user parameter a value. See **setUserParam** for a further discussion of user parameters. See **getErrorInfo** for information about how to detect end-of-list.

Related Functions

getNextUserParam, **setUserParam**

Example

To get the name and value for the first user parameter for the report whose handle is *hRpt*:

```
{
   char param[40], value[100];
   getFirstUserParam (hRpt, (LPSTR)param, 40, (LPSTR)value, 100);
}
```

getHighScope

BOOL FAR PASCAL getHighScope (int *hReport*, **LPSTR** *lpszScope*, **int** *size*);

hReport	Report handle.
lpszScope	High-scope buffer.
size	Size of *lpszScope* buffer.

Return Value

The **getHighScope** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getHighScope** to obtain the current value of the "high scope" parameter for the report specified by *hReport*. **getHighScope** returns the current value as a character string in the buffer pointed to by *lpszScope* to the extent allowed by *size*. In conjunction with the "low scope" and "scope usage" parameters, this parameter can be used to specify the range of master-file records used in printing the report. See **setScopeUsage** for details of how **setHighScope**, **setLowScope**, and **setScopeUsage** interact in providing this capability.

Related Functions

setHighScope, getLowScope, setLowScope, getScopeUsage, setScopeUsage

Example

To get the current scope values for the report whose handle is *hRpt*:

```
{
   char hi[100], lo[100];
   getHighScope (hRpt, (LPSTR)hi, 100);
   getLowScope (hRpt, (LPSTR)lo, 100);
}
```

getLibrary

BOOL FAR PASCAL getLibrary (int *hReport*, **LPSTR** *lpszName*, **int** *size*);

hReport	Report handle.
lpszName	Library-name buffer.
size	Size of *lpszName* buffer.

Return Value

The **getLibrary** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getLibrary** to obtain the current value of the report-library parameter for the report specified by *hReport*. **getLibrary** returns the library name in the buffer pointed to by *lpszName* to the extent allowed by *size*. See **setLibrary** for a discussion of this parameter.

Related Functions

setLibrary

Example

To get the name of the report library for the report whose handle is *hRpt*:

```
{
   getLibrary (hRpt, (LPSTR)lib, 80);
}
```

getLowScope

BOOL FAR PASCAL getLowScope (int *hReport*, **LPSTR** *lpszScope*, **int** *size*);

hReport	Report handle.
lpszScope	Low-scope buffer.
size	Size of *lpszScope* buffer.

Return Value

The **getLowScope** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getLowScope** to obtain the current value of the "low scope" parameter for the report specified by *hReport*. **getLowScope** returns the current value as a character string in the buffer pointed to by *lpszScope* to the extent allowed by *size*. In conjunction with the "high scope" and "scope usage" parameters, this parameter can be used to specify the range of master-file records used in printing the report. See **setScopeUsage** for details of how **setHighScope**, **setLowScope**, and **setScopeUsage** interact in providing this capability.

Related Functions

setLowScope, getHighScope, setHighScope, getScopeUsage, setScopeUsage

Example

To get the current scope values for the report whose handle is *hRpt*:

```
{
   char hi[100], lo[100];
   getHighScope (hRpt, (LPSTR)hi, 100);
   getLowScope (hRpt, (LPSTR)lo, 100);
}
```

getMasterIndexInfo

BOOL FAR PASCAL getMasterIndexInfo (int *hReport*, **LPSTR** *lpszPath*, **int** *pSize*, **LPSTR** *lpszType*, **LPSTR** *lpszTag*, **int** *tagSize*);

hReport	Report handle.
lpszPath	Index-name buffer.
pSize	Size of *lpszPath* buffer.
lpszType	Data-type buffer.
lpszTag	Tag-name buffer.
tagSize	Size of *lpszTag* buffer.

Return Value

The **getMasterIndexInfo** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getMasterIndexInfo** to obtain the current values of the parameters relating to the master index for the report specified by *hReport*. **getMasterIndexInfo** returns the current value of the index file's path and name in the buffer pointed to by *lpszPath* to the extent allowed by *pSize*; the current value of the index file's data type in the buffer pointed to by *lpszType*; and the current value of the index file's tag in the buffer pointed to by *lpszTag* to the extent allowed by *tagSize*. The value returned in *lpszType* is always a single character. See **setMasterIndexInfo** for further details.

Related Functions

setMasterIndexInfo, **setScopeUsage**, **setLowScope**, **setHighScope**

Example

To get information about the master index for the report whose handle is *hRpt*:

```
{
   char path[80], tag[20];
   char type[2];
   getMasterIndexInfo (hRpt, (LPSTR)path, 100, (LPSTR)type,
       (LPSTR)tag, 20);
}
```

getMasterTableName

BOOL FAR PASCAL getMasterTableName (int *hReport*, **LPSTR** *lpszPath*, **int** *pSize*);

hReport	Report handle.
lpszPath	Filename buffer.
pSize	Size of *lpszPath* buffer.

Return Value

The **getMasterTableName** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getMasterTableName** to obtain the current master table name for the report specified by *hReport*. The name is returned in the buffer pointed to by *lpszPath* to the extent allowed by *pSize*.

Related Functions

setMasterTableName

Example

To get the name of the master table for the report whose handle is *hRpt*:

```
{
   char table[80];
   getMasterTableName (hRpt, (LPSTR)table, 80);
}
```

getMemoName

BOOL FAR PASCAL getMemoName (int *hReport*, **LPSTR** *lpszPath*, **int** *pSize*);

hReport	Report handle.
lpszPath	Filename buffer.
pSize	Size of *lpszPath* buffer.

Return Value

The **getMemoName** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getMemoName** to obtain the current ASCII memo filename for the report specified by *hReport*. The name is returned in the buffer pointed to by *lpszPath* to the extent allowed by *pSize*.

Related Functions

setMemoName

Example

To get the name of the ASCII memo file for the report whose handle is *hRpt*:

```
{
   char memo[80];
   getMemoName (hRpt, (LPSTR)memo, 80);
}
```

getNewReportHandle

int FAR PASCAL getNewReportHandle (LPSTR *lpszAppName***);**

lpszAppName	Name of calling application.

Return Value

The **getNewReportHandle** function returns a report-information handle if there are no errors. A return value of zero indicates an error. To obtain more information about the error use **getErrorInfo** with a handle of zero.

Description

Use **getNewReportHandle** to obtain the handle of an empty report-information structure. The *lpszAppName* argument identifies the calling application. This routine is most commonly used (instead of **chooseReport** or **getRuntimeRecord**) when the user will be selecting a report at runtime. See **setReportPick** for a discussion of selection of reports by the user at runtime.

Related Functions

chooseReport, getRuntimeRecord, setReportPick

Example

For a quick way to run a user-selected report without modification:

```
hRpt = getNewReportHandle((LPSTR)"Application Name"); // get a handle
setReportPick (hRpt, 'R');    // let user pick report
execRuntime (hRpt, 0, SW_SHOW, NULL, NULL, NULL, 0);  // run it
```

getNextFieldName

BOOL FAR PASCAL getNextFieldName (int *hReport***, LPSTR** *lpszFieldName***, int** *fnSize***);**

hReport	Report handle.
lpszFieldName	Fieldname buffer.
fnSize	Size of *lpszFieldName* buffer.

Return Value

The **getNextFieldName** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getNextFieldName** in a loop to get the fieldnames available for use in the report specified by *hReport*, after getting the first available fieldname with **getFirstFieldName**. **getNextFieldName** returns the fieldname with alias qualifier in the buffer pointed to by *lpszFieldName* to the extent allowed by *fnSize*. See **getErrorInfo** for information about how to detect end-of-list.

Related Functions

getFirstFieldName, getNextFilteredFieldName

Example

See **getFirstFieldName** for an example of **getNextFieldName**.

getNextFilteredFieldName

BOOL FAR PASCAL getNextFilteredFieldName (**int** *hRepstf*, **LPSTR** *lpszFieldName*, **int** *fnSize*, **int** *filter*);

hReport	Report handle.
lpszFieldName	Fieldname buffer.
fnSize	Size of *lpszFieldName* buffer.
filter	Filter ID.

Return Value

The **getNextFilteredFieldName** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getNextFilteredFieldName** in a loop to get the fieldnames available for use in the report specified by *hReport* suitable for use in the context specified by *filter*, after getting the first such fieldname with **getFirstFieldName**. **getNextFieldName** returns the filename with alias qualifier in the buffer pointed to by *lpszFieldName* to the extent allowed by *fnSize*. See **getErrorInfo** for information about how to detect end-of-list.

The *filter* argument specifies the context to be used in deciding which available fields to return. The valid values for *filter*, defined in rreport.h, are FILTER_ID_SORT and FILTER_ID_GROUP, which return fields suitable for use as sort or group fields, respectively.

Related Functions

getFirstFilteredFieldName, getFirstFieldName, getNextFieldName

Example

See **getFirstFieldName** for an example of adding fieldnames to a combo box. To modify that example to get suitable sort fields, simply change the function names from **getFirstFieldName** and **getNextFieldName** to **getFirstFilteredFieldName** and **getNextFilterFieldName** and add a new last argument to both of FILTER_ID_SORT.

getNextGroupField

BOOL FAR PASCAL getNextGroupField (int *hReport*, **LPSTR** *lpszName*, **int** *nSize*);

hReport	Report handle.
lpszName	Group-field-name buffer.
nSize	Size of *lpszName* buffer.

Return Value

The **getNextGroupField** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getFirstGroupField** and **getNextGroupField** to obtain the current values of the "group field" parameters in the report specified by *hReport*. **getFirstGroupField** returns the name of the first group field in the buffer pointed to by *lpszName*, to the extent allowed by *nSize*. Use **getNextGroupField** iteratively to get the names of the second through eighth group fields. Whenever **getFirstGroupField** is called, the next call to **getNextGroupField** will return the name of the second group field. See **getErrorInfo** for information about how to detect end-of-list.

Related Functions

getFirstGroupField, **setGroupField**, **getFirstSortField**, **getNextSortField**, **setSortField**

Example

See **getFirstGroupField** for an example of **getNextGroupField**.

getNextRelationInfo

BOOL FAR PASCAL getNextRelationInfo (int *hReport*, **LPSTR** *lpszFilePath*, **int** *fSize*, **LPSTR** *lpszIndexPath*, **int** *iSize*, **LPSTR** *lpszTag*, **int** *tSize*, **LPSTR** *lpszAlias*, **int** *aSize*);

hReport	Report handle.
lpszFilePath	Related-filename buffer.
fSize	Size of *lpszFilePath* buffer.
lpszIndexPath	Index-name buffer.
iSize	Size of *lpszIndexPath* buffer.
lpszTag	Tag-name buffer.
tSize	Size of *lpszTag* buffer.
lpszAlias	Alias buffer.
aSize	Size of *lpszAlias* buffer.

Return Value

The **getNextRelationInfo** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getNextRelationInfo** in a loop to obtain information about all related tables but the "first" in the report specified by *hReport*. **getNextRelationInfo** returns the related table's name in the buffer pointed to by *lpszFilePath* to the extent allowed by *fSize*; the index and index tag, if any, used in the relation in the buffers pointed to by *lpszIndexPath* and *lpszTag* to the extent allowed by *iSize* and *tSize*; and the alias of the related table in the

buffer pointed to by *lpszAlias* to the extent allowed by *aSize*. Use **getFirstRelationInfo** to obtain equivalent information about the "first" of the related tables. See **getErrorInfo** for information about how to detect end-of-list.

Related Functions

getFirstRelationInfo, **setRelationInfo**, **getMasterTableName**, **getMasterIndexInfo**

Example

To get information about the next related table in the report whose handle is *hRpt*:

```
{
   char table[80];
   char index[80];
   char tag[20];
   char alias[10];
   getNextRelationInfo (hRpt, (LPSTR)table, 80, (LPSTR)index, 80,
      (LPSTR)tag, 20, (LPSTR)alias, 10);
}
```

This would typically be used in a loop, following a call to **getFirstRelationInfo**.

getNextSortField

BOOL FAR PASCAL getNextSortField (int *hReport*, **LPSTR** *lpszName*, **int** *nSize*);

hReport	Report handle.
lpszName	Sort-field-name buffer.
nSize	Size of *lpszName* buffer.

Return Value

The **getNextSortField** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getFirstSortField** and **getNextSortField** to obtain the current values of the "sort field" parameters in the report specified by *hReport*. **getFirstSortField** returns the name and direction of the first sort field in the buffer pointed to by *lpszName*, to the extent allowed by *nSize*. Use **getNextSortField** iteratively to get the names and directions of the second through eighth sort fields. Whenever **getFirstSortField** is called, the next call to **getNextSortField** will return the name of the second sort field. See **setSortField** for a discussion of the sort field parameters and a description of the syntax of *lpszName*. See **getErrorInfo** for information about how to detect end-of-list.

Related Functions

getFirstSortField, **setSortField**, **getFirstGroupField**, **getNextGroupField**, **setGroupField**

Example

See **getFirstSortField** for an example of **getNextSortField**.

getNextUserParam

BOOL FAR PASCAL getNextUserParam (int *hReport*, **LPSTR** *lpszName*, **int** *nSize*, **LPSTR** *lpszValue*, **int** *vSize*);

hReport	Report handle.
lpszName	Parameter-name buffer.
nSize	Size of *lpszName* buffer.
lpszValue	Parameter-value buffer.
vSize	Size of *lpszValue* buffer.

Return Value

The **getNextUserParam** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getNextUserParam** in a loop to get the names and current values, if any, of the all but the first user parameter for the report specified by *hReport*. The name of the user parameter is returned in the buffer pointed to by *lpszName* to the extent allowed by *nSize*. The current value, if any, is returned in the buffer pointed to by *lpszValue* to the extent allowed by *vSize*. Use **getFirstUserParam** to get the name and value of the first user parameter. Use **setUserParam** to give a user parameter a value. See **setUserParam** for a further discussion of user parameters. See **getErrorInfo** for information about how to detect end-of-list.

Related Functions

getFirstUserParam, **setUserParam**

Example

To get the name and value for the next user parameter for the report whose handle is *hRpt*:

```
{
   char param[40], value[100];

   getNextUserParam (hRpt, (LPSTR)param, 40, (LPSTR)value, 100);
}
```

getOutputDest

BOOL FAR PASCAL getOutputDest (int *hReport*, **LPSTR** *lpszDest*, **int** *dSize*);

hReport	Report handle.
lpszDest	Output-destination buffer.
dSize	Size of *lpszDest* buffer.

Return Value

The **getOutputDest** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getOutputDest** to obtain the current value of the "output destination" parameter for the report specified by *hReport*. **getOutputDest** returns the value in the buffer specified by *lpszDest* to the extent allowed by *dSize*. See **setOutputDest** for a discussion of this parameter.

Related Functions

setOutputDest, **getOutputFile**, **setOutputFile**

Example

To get the current output-destination parameter for a report whose handle is *hRpt*:

```
{
   char dest[30];
   getOutputDest (hRpt, (LPSTR)dest, 30);
}
```

getOutputFile

BOOL FAR PASCAL getOutputFile (int *hReport*, **LPSTR** *lpszName*, **int** *size*);

hReport	Report handle.
lpszName	Output-filename buffer.
size	Size of *lpszName* buffer.

Return Value

The **getOutputFile** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getOutputFile** to obtain the current value of the "output file" parameter for the report specified by hReport. **getOutputFile** returns the value in the buffer specified by *lpszName* to the extent allowed by *size*. See **setOutputFile** for a discussion of this parameter.

Related Functions

setOutputFile, **getOutputDest**, **setOutputDest**

Example

To get the current output file for the report whose handle is *hRpt*:

```
{
   char outfile[80];
   getOutputFile (hRpt, (LPSTR)outfile, 80);
}
```

getPreventEscape

BOOL FAR PASCAL getPreventEscape (int *hReport*, **BOOL FAR *** *lpbNoEsc*);

hReport	Report handle.
lpbNoEsc	Prevent-escape-flag buffer.

Return Value

The **getPreventEscape** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getPreventEscape** to obtain the current setting of the "prevent escape" flag for the report specified by *hReport*. **getPreventEscape** returns this flag value in the buffer pointed to by *lpbNoEsc*. See **setPreventEscape** for a discussion of this flag.

Related Functions

setPreventEscape

Example

To get the prevent-escape flag for the report whose handle is *hRpt*:

```
{
   BOOL noEscape;
   getPreventEscape (hRpt, (BOOL FAR *)&noEscape);
}
```

getPrinter

BOOL FAR PASCAL getPrinter (int *hReport*, **LPSTR** *lpszPrinter*, **int** *size*);

hReport	Report handle.
lpszPrinter	Printer-name buffer.
size	Size of *lpszPrinter* buffer.

Return Value

The **getPrinter** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getPrinter** to obtain the current value of the "printer" parameter for the report specified by *hReport*. **getPrinter** returns the value in the buffer pointed to by *lpszPrinter* to the extent allowed by *size*. See **setPrinter** for a discussion of this parameter.

Related Functions

setPrinter, **getPrinterPort**, **setPrinterPort**

Example

To get the current printer parameter for the report whose handle is *hRpt*:

```
{
   char printer[100];
   getPrinter (hRpt, (LPSTR)printer, 100);
}
```

getPrinterPort

BOOL FAR PASCAL getPrinterPort (int *hReport*, **LPSTR** *lpszPort*, **int** *size*);

hReport	Report handle.
lpszPort	Printer-port-name buffer.
size	Size of *lpszPort* buffer.

Return Value

The **getPrinterPort** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getPrinterPort** to obtain the current value of the "printer port" parameter for the report specified by *hReport*. **getPrinterPort** returns the value in the buffer pointed to by *lpszPort* to the extent allowed by *size*. See **setPrinterPort** for a discussion of this parameter.

Related Functions

setPrinterPort, **getPrinter**, **setPrinter**

Example

To get the current printer-port parameter for the report whose handle is *hRpt*:

```
{
   char port[10];
   getPrinterPort (hRpt, (LPSTR)port, 10);
}
```

getReportPick

BOOL FAR PASCAL getReportPick (int *hReport*, **LPSTR** *lpszPickFlag*);

hReport	Report handle.
lpszPickFlag	Report-selection-flag buffer.

Return Value

The **getReportPick** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getReportPick** to obtain the current value of the report-selection parameter for the report specified by *hReport*. **getReportPick** returns the current value of this parameter in the form of a single character in the buffer pointed to by *lpszPickFlag*. See **setReportPick** for a discussion of this flag.

Related Functions

setReportPick

Example

To get the report-selection parameter for the report whose handle is *hRpt*:

```
{
   char pick[2];
   getReportPick (hRpt, (LPSTR)pick);
}
```

getRuntimeRecord

int FAR PASCAL getRuntimeRecord (LPSTR *lpszAppName***, LPSTR** *lpszControlFile***);**

lpszAppName	Name of calling application.
lpszControlFile	Pointer to ASCII runtime control filename.

Return Value

The **getRuntimeRecord** function returns a report-information handle if there are no errors. A return value of zero indicates an error. To obtain more information about the error use **getErrorInfo** with a report handle of zero.

Description

Use **getRuntimeRecord** to begin processing a report based on information in the ASCII Viewer Control File whose name is pointed to by *lpszControlFile*. The control file pointed to by *lpszControlFile* must contain a non-empty value for RI_REPORT and may also contain a non-empty value for RI_LIBRARY. If both are non-empty, RI_LIBRARY is treated as the name of a report library and RI_REPORT is treated as the name of a report within that library. If only RI_REPORT is non-empty, it is treated as the name of a report file. The *lpszAppName* argument identifies the calling application. The handle returned by **getRuntimeRecord** is used as input to most other functions contained within this API.

Related Functions

chooseReport, getNewReportHandle, writeRuntimeRecord

Example

To read an existing ASCII Viewer Control File, modify some parameters, and then save the results in the same file:

```
{
   int hRpt;
   if (hRpt = getRuntimeRecord ((LPSTR)"App Name",
      (LPSTR)"c:\\rrdata\\runrecd"))
   {
      setScopeUsage (hRpt, 'E');
      setFilterUsage (hRpt, 'E');
      writeRuntimeRecord (hRpt, NULL);
   }
}
```

getScopeUsage

BOOL FAR PASCAL getScopeUsage (int *hReport*, **LPSTR** *lpszScopeFlag*);

hReport	Report handle.
lpszScopeFlag	Scope-usage-flag buffer.

Return Value

The **getScopeUsage** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getScopeUsage** to obtain the current value of the "scope usage" parameter for the report specified by *hReport*. **getScopeUsage** returns the current value of this parameter in the form of a single character in the buffer pointed to by *lpszScopeFlag*. See **setScopeUsage** for a discussion of scope-usage values and the interaction among values set by **setScopeUsage**, **setLowScope**, and **setHighScope**.

Related Functions

setScopeUsage, getLowScope, setLowScope, getHighScope, setHighScope

Example

To read the scope-usage flag for the report whose handle is *hRpt*:

```
{
   char scopeUsage[2];
   getScopeUsage (hRpt, (LPSTR)scopeUsage);
}
```

getStatusEveryPage

BOOL FAR PASCAL getStatusEveryPage (int *hReport*, **BOOL FAR *** *lpbStatus*);

hReport	Report handle.
lpbStatus	Status-frequency buffer.

Return Value

The **getStatusEveryPage** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getStatusEveryPage** to obtain the current value of the "status every page" parameter for the report specified by *hReport*. **getStatusEveryPage** returns the current value of this parameter in the form of a boolean in the buffer pointed to by *lpbStatus*. See **setStatusEveryPage** for a further description of this parameter.

Related Functions

setStatusEveryPage

Example

To get the status-every-page flag for the report whose handle is *hRpt*:

```
{
   BOOL pageStatus;
   getStatusEveryPage (hRpt, (BOOL FAR *)&pageStatus);
}
```

getTestPattern

BOOL FAR PASCAL getTestPattern (int *hReport*, **BOOL FAR *** *lpbTest*);

hReport	Report handle.
lpbTest	Test-pattern-flag buffer.

Return Value

The **getTestPattern** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getTestPattern** to obtain the current value of the "test pattern" parameter for the report specified by *hReport*. **getTestPattern** returns the current value of this parameter in the form of a boolean in the buffer pointed to by *lpbTest*. See **setTestPattern** for a further description of this parameter.

Related Functions

setTestPattern

Example

To get the test-pattern flag for the report whose handle is *hRpt*:

```
{
   BOOL test;
   getTestPattern (hRpt, (BOOL FAR *)&test);
}
```

getWinTitle

BOOL FAR PASCAL getWinTitle (int *hReport*, **LPSTR** *lpszTitle*, **int** *size*);

hReport	Report handle.
lpszTitle	Report-title buffer.
size	Size of *lpszTitle* buffer.

Return Value

The **getWinTitle** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **getWinTitle** to obtain the current value of the "report title" parameter for the report specified by *hReport*. **getWinTitle** returns the title in the buffer pointed to by *lpszTitle* to the extent allowed by *size*. See **setWinTitle** for a discussion of the report title parameter.

Related Functions

setWinTitle

Example

To get the current report-title string for the report whose handle is *hRpt*:

```
{
   char title[100];
   getWinTitle (hRpt, (LPSTR)title, 100);
}
```

resetErrorInfo

BOOL FAR PASCAL resetErrorInfo (int *hReport***);**

Return Value

The **resetErrorInfo** function always returns non-zero.

Description

Use **resetErrorInfo** to force the runtime DLL to reset its error information variables for the report indicated by *hReport*. The error message and code returned by **getErrorInfo** always pertain to calls made since the last call to **resetErrorInfo** for the specified report.

Related Functions

getErrorInfo

Example

To reset the error information:

```
resetErrorInfo();
```

setBeginPage

BOOL FAR PASCAL setBeginPage (int *hReport***, LONG** *lBeginPage***);**

hReport	Report handle.
lBeginPage	Starting page number.

Return Value

The **setBeginPage** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setBeginPage** to replace the current value of the "starting page" parameter for the report specified by *hReport* with the value specified by *lBeginPage*. The "starting page" parameter can be used to override the starting page number saved with the report. One application for this parameter is for restarting a canceled report without reprinting the parts that were already printed. See **execRuntime** for a discussion of how to restart a partially printed report. Be sure that the value specified with **setBeginPage** is no larger than the one specified with **setEndPage**.

Related Functions

getBeginPage, **setEndPage**, **getEndPage**, **setStatusEveryPage**, **execRuntime**

Example

To print pages 10 to 15 of the report whose handle is *hRpt*:

```
setBeginPage (hRpt, 10L);
setEndPage (hRpt, 15L);
```

setCopies

BOOL FAR PASCAL setCopies (int *hReport*, **int** *copies*);

hReport	Report handle.
copies	Number of copies.

Return Value

The **setCopies** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setCopies** to replace the current value of the "number of copies" parameter for the report specified by *hReport* with the value specified by *copies*. The specified value must be between 0 and 999, inclusive. A value of 0 causes Viewer to revert to the number of copies saved with the report.

Related Functions

getCopies

Example

To set the number of copies for the report whose handle is *hRpt* to 2:

```
setCopies (hRpt, 2);
```

setDataDir

BOOL FAR PASCAL setDataDir (int *hReport*, **LPSTR** *lpszDir*);

hReport	Report handle.
lpszDir	Default data directory.

Return Value

The **setDataDir** function returns zero if an error occurs. To obtain more information about the error use getErrorInfo.

Description

Use **setDataDir** to replace the default data directory specified in RRW.INI with the value specified by *lpszDir*, for the report specified by *hReport*. Viewer may use the default data directory in trying to locate tables used in the report specified by *hReport*.

Related Functions

setImageDir, setLibraryDir

Example

To specify the use of c:\rrdata as the default data directory for the report whose handle is *hRpt*:

```
setDataDir (hRpt, (LPSTR)"c:\\rrdata");
```

setDisplayErrors

BOOL FAR PASCAL setDisplayErrors (int *hReport*, **BOOL** *bDisperr*);

hReport	Report handle.
bDisperr	Display-errors flag.

Return Value

The **setDisplayErrors** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setDisplayErrors** to replace the current value of the "display errors" flag for the report specified by *hReport* with the value specified by *bDisperr*. If the "display errors" flag is non-zero, error messages will be displayed on the screen, in addition to being returned to the calling application; otherwise, error messages are only returned to the calling application. Error messages are returned to the calling application via the *lpszEMsg* buffer supplied to **execRuntime** or via the RO_EMSG field in the Viewer Status File, depending on the value of *bWait* passed to **execRuntime**. By default, error messages are not displayed on the screen.

Related Functions

getDisplayErrors, execRuntime

Example

To specify that, for the report whose handle is *hRpt*, Viewer should display errors as well as return them:

```
setDisplayErrors (hRpt, 1);
```

setDisplayStatus

BOOL FAR PASCAL setDisplayStatus (int *hReport***, BOOL** *bDispStatus***);**

hReport	Report handle.
bDispstatus	Display-status flag.

Return Value

The **setDisplayStatus** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setDisplayStatus** to replace the current value of the "display status" flag for the report specified by *hReport* with the value specified by *bDispStatus*. If the "display status" flag is non-zero, a status window is displayed while the report is being generated; otherwise it will display an icon while it is running. By default, status is not displayed. If "display status" is non-zero and the "prevent escape" flag is zero, the status window will contain a Cancel button that will allow the user to terminate a report in progress.

Related Functions

getDisplayStatus, **setPreventEscape**, **getPreventEscape**

Example

To specify that, for the report whose handle is *hRpt*, Viewer should display a status window, and that the window should include a Cancel button:

```
setDisplayStatus (hRpt, 1);   // display a status window...
setPreventEscape (hRpt, 0);   // ... with a Cancel button
```

setEndPage

BOOL FAR PASCAL setEndPage (int *hReport***, LONG** *lEndPage***);**

hReport	Report handle.
lEndPage	Ending page number.

Return Value

The **setEndPage** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setEndPage** to replace the current value of the "ending page" parameter for the report specified by *hReport* with the value specified by *lEndPage*. The "ending page" parameter can be used to override the ending page number saved with the report. Be sure that the value specified by **setEndPage** is at least as large as the value specified by **setBeginPage**.

Related Functions

getEndPage, setBeginPage, getBeginPage

Example

To print pages 10 to 15 of the report whose handle is *hRpt*:

```
setBeginPage (hRpt, 10L);
setEndPage (hRpt, 15L);
```

setExportDest

BOOL FAR PASCAL setExportDest (int *hReport*, **char** *cVal*);

hReport	Report handle.
c*Val*	Export-destination flag.

Return Value

The **setExportDest** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setExportDest** to replace the current value of the "export destination" parameter for the report specified by *hReport* with the value specified by *cVal*. The export destination is used to specify how the results of an Excel Chart or Excel PivotTable export are to be presented. Valid values for this parameter are:

- **D** (Display) means to present the results of the PivotTable or Chart export on the display from within Excel.
- **F** (File) means to save the PivotTable or Chart export to the file specified by **setOutputFile**.
- **P** (Printer) means to print the PivotTable or Chart to Excel's default printer.

Related Functions

getExportDest, setOutputFile

Example

To indicate that the PivotTable or Chart report whose handle is *hRpt* should be displayed by Excel:

```
setExportDest (hRpt, 'D');
```

setFilter

BOOL FAR PASCAL setFilter (int *hReport*, **LPSTR** *lpszFilter*);

hReport	Report handle.
lpszFilter	Filter expression.

Return Value

The **setFilter** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setFilter** to specify a filter expression, *lpszFilter*, that may be used instead of the filter, if any, saved with the report specified by *hReport*. Viewer will use this filter expression only if you also call **setFilterUsage** with a value of **O**. See **setFilterUsage** for details of this behavior. A filter expression must use the same syntax as that of a calculated field expression that returns a logical value. The expression can include any database, calculated, or total fields available in the report, along with built-in function references, constants, and UDF references. When Viewer uses the expression specified via **setFilter**, it will include only those records where the value of the expression is true. The maximum size of a filter expression is 1024.

Related Functions

setFilterUsage, **getFilter**, **getFilterUsage**

Example

To limit the data of the report whose handle is *hRpt* to those records where CITY is Boston or Westborough and STATE is MA:

```
setFilter(hRpt, (LPSTR)"STATE='MA' AND
   (CITY='Boston' OR CITY='Westborough')");
setFilterUsage (hRpt, 'O');   // override saved filter
```

Note the use of parentheses in the filter expression. Without the parentheses, the filter would accept a CITY value of Westborough even if the STATE were not MA, since RRW gives AND and OR equal precedence and evaluates them from left-to-right.

setFilterUsage

BOOL FAR PASCAL setFilterUsage (int *hReport*, **char** *cVal*);

hReport	Report handle.
c*Val*	Filter-usage flag.

Return Value

The **setFilterUsage** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setFilterUsage** to set the "filter usage" parameter for the report specified by *hReport* to the value specified by *cVal*. Valid values for this parameter are:

- **S** (Saved) means to run the report using the filter saved with it, if any. Viewer will ignore any expression specified via **setFilter** and run the report exactly as it was saved.
- **E** (Entire) means to ignore any filter saved in the report or specified via **setFilter**.

- **O** (Override) means to override the saved filter, if any, with the expression specified via **setFilter**.
- **?** (Question mark) means to allow the user to enter a filter or edit the saved filter at runtime. If no filter was saved with the report, the Insert Selection Rule dialog displays, as shown in Figure 3.1.

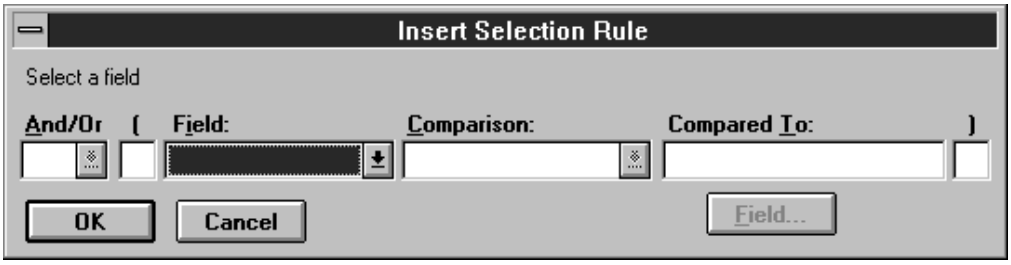

Figure 3.1 Insert Selection Rule Dialog Box

If a filter was saved with the report, the Query dialog box displays, as shown in Figure 3.2.

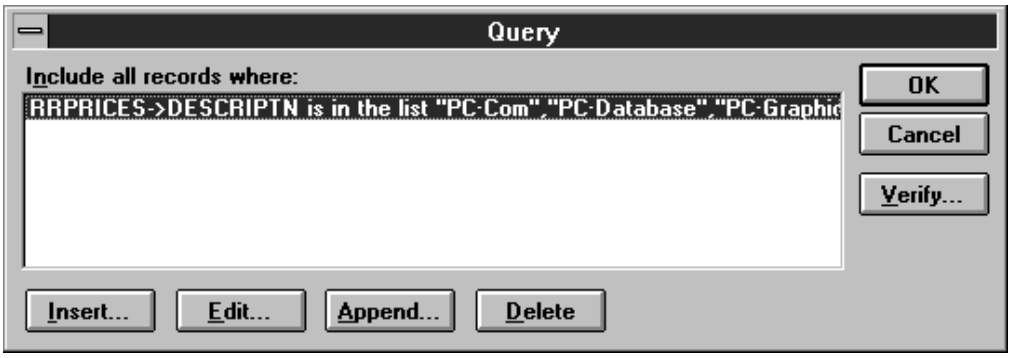

Figure 3.2 Query Dialog Box

When the filter-usage flag is a question mark (**?**), the value specified via **setFilter** is always ignored.

Related Functions

setFilter, **getFilter**, **getFilterUsage**

Example

To allow the user to specify a filter at runtime for the report whose handle is *hRpt*:

```
setFilterUsage (hRpt, '?');
```

setGroupField

BOOL FAR PASCAL setGroupField (int *hReport*, **LPSTR** *lpszName*, **int** *groupNum*);

hReport	Report handle.
lpszName	Group-field name.
groupNum	Group number.

Return Value

The **setGroupField** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setGroupField** to replace an existing group field or add a new one to the report specified by *hReport*. Pass the group field number to be added or replaced in *groupNum* and its name in *lpszName*. You must replace all group fields from group field 1 through the last group field you wish to replace. For example, if you only wish to replace group field 2, you must call **setGroupField** twice, once with a *groupNum* of 1 and once with a *groupNum* of 2. To obtain the current group field parameters, use **getFirstGroupField** and **getNextGroupField**.

Related Functions

getFirstGroupField, **getNextGroupField**, **setSortField**, **getFirstSortField**, **getNextSortField**

Example

To replace the second group field with CITY, while leaving the first group field unchanged for the report whose handle is *hRpt*:

```
{
   char buf[80];
   getFirstGroupField (hRpt, (LPSTR)buf, 80);
   setGroupField (hRpt, (LPSTR)buf, 1);
   setGroupField (hRpt, (LPSTR)"CITY", 2);
}
```

setHighScope

BOOL FAR PASCAL setHighScope (int *hReport*, **LPSTR** *lpszScope*);

hReport	Report handle.
lpszScope	High-scope value.

Return Value

The **setHighScope** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setHighScope** to set the "high scope" parameter for the report specified by *hReport* to the value specified by *lpszScope*. See **setScopeUsage** for details.

Related Functions

getHighScope, **setLowScope**, **getLowScope**, **setScopeUsage**, **getScopeUsage**

Example

Assuming we have an index on the NAME field of the master table, to limit the records from the master table to those where the NAME field begins with a letter between **A** and **M**, inclusive, for the report whose handle is *hRpt*:

```
setScopeUsage (hRpt, 'O');    // override saved scope
setMasterIndexInfo (hRpt, (LPSTR)"c:\\data\\name.ndx", 'C', NULL);
setLowScope (hRpt, (LPSTR)"A");
setHighScope (hRpt, (LPSTR)"M");
```

setImageDir

BOOL FAR PASCAL setImageDir (int *hReport*, **LPSTR** *lpszDir*);

hReport	Report handle.
lpszDir	Default image directory.

Return Value

The **setImageDir** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setImageDir** to replace the default image directory specified in RRW.INI with the value specified by *lpszDir*, for the report specified by *hReport*. Viewer may use the default image directory in trying to locate images used in the report specified via *hReport*.

Related Functions

setDataDir, setLibraryDir

Example

To specify the use of c:\rrdata as the default image directory for the report whose handle is *hRpt*:

```
setImageDir (hRpt, (LPSTR)"c:\\rrdata");
```

setIndexExtension

BOOL FAR PASCAL setIndexExtension (int *hReport*, **int** *extNumber*);

hReport	Report handle.
extNumber	Extension index.

Return Value

The **setIndexExtension** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setIndexExtension** to replace the current value of the default-index-extension parameter for the report specified by *hReport* with the value specified by *extNumber*. The default index extension is used by the Viewer executable to locate indexes that are specified without extensions or that it is unable to find using the extensions saved with the report.

The possible values and meanings for *extNumber* are:

0	none
1	CDX
2	IDX
3	MDX
4	NDX
5	NSX
6	NTX
7	WDX

Related Functions

setMasterIndexInfo, setRelationInfo

Example

To set the default index extension for the report specified by *hRpt* to NDX:

```
setIndexExtension (hRpt, 4);
```

setLibrary

BOOL FAR PASCAL setLibrary (int *hReport*, **LPSTR** *lpszName*);

hReport	Report handle.
lpszName	Library-name buffer.

Return Value

The **setLibrary** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setLibrary** to replace the current value of the report-library parameter for the report specified by *hReport* with the value specified by *lpszName*. It is not necessary to call **setLibrary** after obtaining a report handle with **chooseReport** or **getRuntimeRecord** since both of these routines imply the selection of a report library. This routine is primarily for use with **getNewReportHandle** and **setReportPick**.

If *lpszName* does not include a path, the Viewer looks for the library in the directory specified by **setLibraryDir**. If **setLibraryDir** has not been called, the Viewer looks in the default report/library directory specified in RRW.INI. If no default is specified in the INI file either, the Viewer looks for the library in the current directory.

Related Functions

getLibrary, **getNewReportHandle**, **setReportPick**, **setLibraryDir**

Example

To specify the library c:\libs\acctrpts for a report-information handle obtained via a call to **getNewReportHandle**, allowing the user to pick a single report to run:

```
{
   char emsg[256];
   int ecode;
   long pgct;
   int hRpt = getNewReportHandle();
   if (hRpt)
   {
      if (setLibrary (hRpt, (LPSTR)"c:\\libs\\acctrpts"));
      {
         setReportPick (hRpt, 'R');
         execRuntime (hRpt, 1, SW_SHOW, (LPINT)&ecode,
            (LPLONG)&pgct, (LPSTR)emsg, 256);
      }
      else ... // error handling
   }
   else ...    // error handling
}
```

setLibraryDir

BOOL FAR PASCAL setLibraryDir (int *hReport***, LPSTR** *lpszDir***);**

hReport	Report handle.
lpszDir	Default report or library directory.

Return Value

The **setLibraryDir** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setLibraryDir** to replace the default report/library directory specified in RRW.INI with the value specified by *lpszDir*, for the report specified by *hReport*. Viewer may use the default report/library directory in trying to locate the library or report specified via **setLibrary** or **chooseReport,** or implicitly via **getRuntimeRecord**.

Related Functions

setDataDir, setImageDir, setLibrary, chooseReport, getRuntimeRecord

Example

To specify the use of c:\rrdata as the default library directory for the report whose handle is *hRpt*:

```
setLibraryDir (hRpt, (LPSTR)"c:\\rrdata");
```

setLowScope

BOOL FAR PASCAL setLowScope (int *hReport*, **LPSTR** *lpszScope*);

hReport	Report handle.
lpszScope	Low-scope value.

Return Value

The **setLowScope** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setLowScope** to set the "low scope" parameter for the report specified by *hReport* to the value specified by *lpszScope*. See **setScopeUsage** for details.

Related Functions

getLowScope, **setHighScope**, **getHighScope**, **setScopeUsage**, **getScopeUsage**

Example

Assuming we have an index on the NAME field of the master table, to limit the records from the master table to those where the NAME field begins with a letter between **A** and **M**, inclusive, for the report whose handle is *hRpt*:

```
setScopeUsage (hRpt, 'O');    // override saved scope
setMasterIndexInfo (hRpt, (LPSTR)"c:\\data\\name.ndx", 'C', NULL);
setLowScope (hRpt, (LPSTR)"A");
setHighScope (hRpt, (LPSTR)"M");
```

setMasterIndexInfo

BOOL FAR PASCAL setMasterIndexInfo (int *hReport*, **LPSTR** *lpszPath*, **char** *cType*, **LPSTR** *lpszTag*);

hReport	Report handle.
lpszPath	Index-name buffer.
cType	Data type.
lpszTag	Tag-name buffer.

Return Value

The **setMasterIndexInfo** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setMasterIndexInfo** to set the master index parameters for the report specified by *hReport*. Use *lpszPath* to specify the path and/or filename of a master index; *cType* to indicate the data type of the index; and *lpszTag* to specify the master index tag. For *lpszPath*:

- If you specify both a directory and a file name, this directory is the only directory searched and this file name is the only file the Viewer searches for.
- If you specify a directory without a file name, the Viewer searches the specified directory for the master index name saved with the report.
- If you specify a file name without a directory, the Viewer searches for a file with the specified name in the directory of the master index saved with the report, then in the current master database directory, then in the default data directory specified via **setDataDir** or in RRW.INI. If no default is specified, the Viewer searches for the file in the current directory.
- If *lpszPath* is NULL or points to a null string, the Viewer will use the saved index unless *cType* is **R**, in which case you are removing a master index saved with the report.

The *cType* argument specifies the data type of the new index, where **N** indicates numeric, **D** indicates date, and **C** indicates character. If *cType* is a space character, Viewer will assume that the new index has the same data type as the saved master index, though it is good practice to include an explicit data type specifier. If the index named by *lpszPath* is a multiple-field index file (MDX, CDX, or WDX), *lpszTag* specifies an index tag.

You can also use **setMasterIndexInfo** to remove a master index saved with a report, by specifying *lpszPath* and *lpszTag* as NULL (or the null string), and specifying *cType* as **R**.

Related Functions

getMasterIndexInfo, **setMasterTableName**, **getMasterTableName**

Example

To specify the use of the date tag HIREDATE of the index C:\DATA\EMPLOY.MDX for the report whose handle is *hRpt*:

```
setMasterIndexInfo (hRpt, (LPSTR)"c:\\data\\employ.mdx", 'D',
   (LPSTR)"HIREDATE");
```

setMasterTableName

BOOL FAR PASCAL setMasterTableName (int *hReport*, **LPSTR** *lpszTable*);

hReport	Report handle.
lpszTable	Name buffer.

Return Value

The **setMasterTableName** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setMasterTableName** to replace the master table saved with the report specified by *hReport* with the master table specified by *lpszTable*. The fields in the master table specified by *lpszTable* must match in name, number, and type those in the original master table.

Example

To specify the use of the table C:\DATA\EMPLOY.DBF, for the report whose handle is *hRpt*:

```
setMasterTableName (hRpt, (LPSTR)"c:\\data\\employ.dbf");
```

setMemoName

BOOL FAR PASCAL setMemoName (int *hReport***, LPSTR** *lpszPath***);**

hReport	Report handle.
lpszPath	Pathname buffer.

Return Value

The **setMemoName** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setMemoName** to replace the ASCII memo file used in the report specified by *hReport* with the file specified by *lpszPath*.

- If *lpszPath* specifies both a directory and a table name, this directory is the only directory searched and this file name is the only file the Viewer searches for.
- If *lpszPath* specifies a directory without a file name, the Viewer searches the specified directory for the ASCII memo file name saved with the report.
- If *lpszPath* specifies a file name without a directory, the Viewer searches for a file with the specified name in the directory of the ASCII memo file saved with the report, then in the default data directory specified via **setDataDir** or in RRW.INI. If no default is specified via **setDataDir**, the Viewer searches for the specified table in the current directory.

Related Functions

getMemoName

Example

To specify the use of the ASCII memo file, C:\DATA\LETTER.TXT for the report whose handle is *hRpt*:

```
setMemoName (hRpt, (LPSTR)"c:\\data\\letter.txt");
```

setOutputDest

BOOL FAR PASCAL setOutputDest (int *hReport*, **LPSTR** *lpszDest*);

hReport	Report handle.
lpszDest	Output destination buffer.

Return Value

The **setOutputDest** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setOutputDest** to replace the current value of the "output destination" parameter for the report specified by *hReport*. If you don't call **setOutputDest**, Viewer will print to the destination saved with the report (or to the printer specified via **setPrinter** function). This parameter can have one of the following values: **D, A, T, P, Excel Chart, Excel PivotTable, CSV, MSWORD, RTF, W, X,** or a question mark (**?**).

- A value of **D** specifies that the report be sent to the display, allowing the user to preview the report before printing it. After previewing the report, the user can select Print on the Preview screen to send the report to the printer saved with the report or specified via the **setPrinter** function. Note that if the value of *lpszDest* is **D** and a filename has been specified via **setOutputFile**, the report will be output to the file specified via **setOutputFile** when the user selects Print in Preview.
- A value of **A** or **T** specifies that the report be sent to the text file named via the **setOutputFile** function. The report will be exported as a text file without printer codes.
- A value of **P** specifies that the report be sent to the printer saved with the report or specified via **setPrinter**, even if the report's saved destination is a file.
- A value of **Excel Chart** or **Excel PivotTable** specifies that the report be exported to an Excel chart or PivotTable, respectively. You can use this in conjunction with **setExportDest** to control the export destination (display, file, or printer).
- A value of **CSV, MSWORD,** or **RTF** specifies that the report be exported to a text data file, Word Merge file, or Rich Text Format file, respectively, using either the saved file name or the file name specified via **setOutputFile**.
- A value of **W** specifies that the report be exported to a worksheet file whose name is specified via **setOutputFile**.
- A value of **X** specifies that the report be exported to an Xbase file whose name is specified via **setOutputFile**.
- A value of question mark (**?**) allows the user to select the print destination (screen or printer) at runtime. When the value of *lpszDest* is a question

mark, the user will see the dialog box shown in Figure 3.3. If a title has been specified via **setWinTitle**, the title bar will contain that title; otherwise, the title bar will contain the report name.

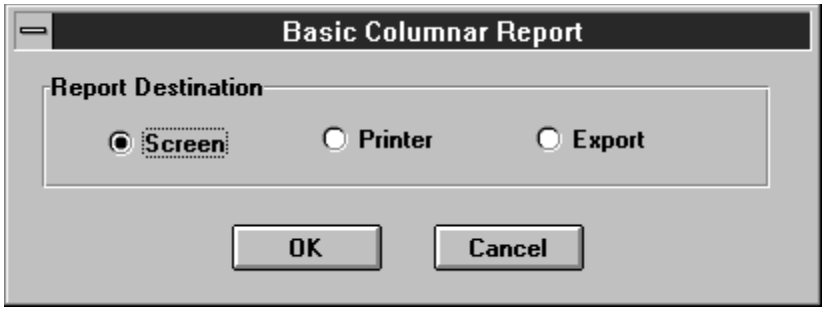

Figure 3.3 Print Destination Dialog Box

The user can select Screen to preview the report, Printer to print it, or Export to export it to one of the available export types (Excel PivotTable, Excel Chart, Rich Text Format, Text, Text Data, Word Merge, Xbase, or Worksheet). If the user selects Cancel, the report will not run and the "Canceled" message will be returned as report status.

If you call neither **setOutputDest** nor **setOutputFile**, the Viewer outputs the report to the printer saved with the report or specified via **setPrinter**. If you call **setOutputFile** but not **setOutputDest**, the Viewer outputs the report to the specified file with printer codes for the printer saved with the report or specified via **setPrinter**.

Related Functions

getOutputDest, setOutputFile, getOutputFile, setPrinter, getPrinter

Example

To specify the display as the output destination for the report whose handle is *hRpt*:

```
setOutputDest (hRpt, "D");
```

setOutputFile

BOOL FAR PASCAL setOutputFile (int *hReport***, LPSTR** *lpszName***);**

hReport	Report handle.
lpszName	Output filename.

Return Value

The **setOutputFile** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setOutputFile** to replace the current value of the "output file" parameter for the report specified by *hReport* with the value specified by *lpszName*. Use it to save report output as a file for printing later, or use it in conjunction with **setOutputDest** to export a report to a file. When this parameter is specified and **setOutputDest** has not been called or has been used to specify a value of **D** or question mark (**?**), the report will be output to a file with printer codes. When this parameter is specified and **setOutputDest** has been used to specify a value of **A**, the report will be output as a text file without printer codes. To send the report directly to the saved destination, do not call **setOutputFile** or **setOutputDest**.

The name of the output file can include a path. For example, to send a report to a text file INVOICE.TXT in the C:\PROJECT\TEXT subdirectory, specify the following value for the *lpszName* parameter:

C:\PROJECT\TEXT\INVOICE.TXT

If *lpszName* does not include a path, the Viewer places the file in the current directory.

Related Functions

getOutputFile, **setOutputDest**, **getOutputDest**

Example

To specify C:\TEMP\REPORT.TXT as the output file for the report whose handle is *hRpt*:

```
setOutputFile (hRpt, (LPSTR)"c:\\temp\\report.txt");
```

setPreventEscape

BOOL FAR PASCAL setPreventEscape (int *hReport*, **BOOL** *bNoEsc*);

hReport	Report handle.
bNoEsc	Prevent-escape flag.

Return Value

The **setPreventEscape** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setPreventEscape** to specify whether or not the user should be able to terminate the report specified by *hReport*. If *bNoEsc* is true, the user will not be able to terminate the report while Viewer is generating it. A value of zero means that a Cancel button will appear in the status window, enabling the user to pause or cancel the report. Note that a status window will appear only if **setDisplayStatus** has been called with a non-zero value. The default value of the "prevent escape" flag is zero. If the user cancels the report, the error-code value returned via *lpiECode* from **execRuntime** or as RO_ECODE in the runtime status file will be **C**.

Related Functions

getPreventEscape, setDisplayStatus, getDisplayStatus, execRuntime

Example

To specify that, for the report whose handle is *hRpt*, Viewer should display a status window and that the window should include a Cancel button:

```
setDisplayStatus (hRpt, 1);   // display a status window...
setPreventEscape (hRpt, 0);   // ... with a Cancel button
```

setPrinter

BOOL FAR PASCAL setPrinter (int *hReport*, **LPSTR** *lpszPrinter*);

hReport	Report handle.
lpszPrinter	Printer name.

Return Value

The **setPrinter** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setPrinter** to replace the current value of the "printer" parameter for the report specified by *hReport* with the printer name specified by *lpszPrinter*. This parameter can have one of these values:

- The name of an available Windows printer (for example, "HP LaserJet Series III"). The value is case insensitive (that is, you can enter it in upper, lower, or mixed case).
- The question mark (**?**) value, to allow the user to select a printer at runtime. When the *lpszPrinter* value is a question mark, the Print dialog will display, as shown in Figure 3.4.
- The word "Default" to force the Viewer to use the current default Windows printer. Use this setting only if you are sure that the default printer is compatible with the layout of your report(s)

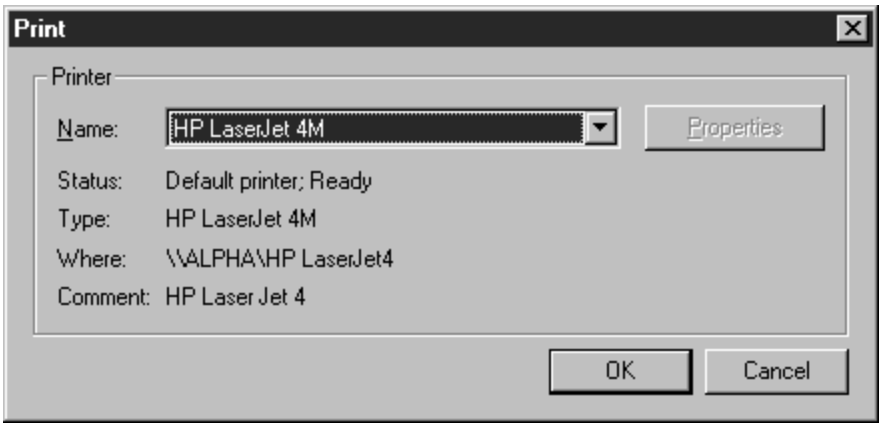

Figure 3.4 Print Dialog Box

The Printers applet (accessible from the Windows Control Panel) controls which printers are listed in the Print dialog box. Viewer initially selects the printer saved with the report. The user can select another printer and port as necessary.

Related Functions

getPrinter, **setPrinterPort**, **getPrinterPort**, **setOutputDest**, **getOutputDest**

Example

To allow the user to select a printer interactively in Viewer for the report whose handle is *hRpt*:

```
setPrinter (hRpt, (LPSTR)"?");
```

setPrinterPort

BOOL FAR PASCAL setPrinterPort (int *hReport***, LPSTR** *lpszPort***);**

hReport	Report handle.
lpszPort	Printer-port name.

Return Value

The **setPrinterPort** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setPrinterPort** to replace the value of the "printer port" parameter for the report specified by *hReport* with the value specified by *lpszPort*. Enter a value such as "LPT1:" to override the current printer port value. Note that the colon is required.

You can also use the question mark (**?**) value or enter the word "Default" for this parameter. When the value of *lpszPort* is a question mark, the user will see the Print dialog

box shown in Figure 3.4. When the value of *lpszPort* is "Default," Viewer will use the default Windows printer port. (See the description of the **setPrinter** function.)

Related Functions

getPrinterPort, **setPrinter**, **getPrinter**, **setOutputDest**, **getOutputDest**

Example

To allow the user to select a printer interactively in Viewer for the report whose handle is *hRpt*:

```
setPrinterPort (hRpt, (LPSTR)"?");
```

setRelationInfo

BOOL FAR PASCAL setRelationInfo (int *hReport*, **LPSTR** *lpszFilePath*, **LPSTR** *lpszIndexPath*, **LPSTR** *lpszTag*, **LPSTR** *lpszAlias*, **int** *aliasNum*);

hReport	Report handle.
lpszFilePath	Related file name.
lpszIndexPath	Index file name.
lpszTag	Tag name.
lpszAlias	Alias.
aliasNum	Relation-override number.

Return Value

The **setRelationInfo** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setRelationInfo** to replace the parameters of a related table in the report specified by *hReport*. Use *lpszFilePath* to specify the new related filename, *lpszIndexPath* and *lpszTag* to specify the index and index tag, if any, to be used to access the new related file, and *lpszAlias* to specify the alias of the related file being replaced. Any argument that is the NULL pointer signifies to leave that particular relation information unchanged. Use an *aliasNum* between 1 and 99 to identify which alias parameter is to be used for the replacement.

Related Functions

getFirstRelationInfo, **getNextRelationInfo**, **setMasterTableName**, **getMasterTableName**

Example

Suppose the report specified by *hReport* includes a related file c:\data\fy93.dbf, whose alias is fy and whose data is scanned via the month tag in the index c:\data\fy93.mdx. If you wish to use **setRelationInfo** to replace fy93.dbf with fy94.dbf and fy93.mdx with fy94.mdx, you might call **setRelationInfo** as follows:

```
setRelationInfo (hReport,          // report handle
    (LPSTR)"c:\\data\\fy94",          // dbf name
    (LPSTR)"c:\\data\\fy94",          // index name
    (LPSTR)"",                    // tag
    (LPSTR)"fy",                  // alias
    1);                           // number
```

which uses an *aliasNum* of 1 to replace fy93 data with fy94 data. Note that the *lpszAlias* value must match the alias of the related file as saved with the report—"fy". Also, note that the *lpszTag* value is a null string, which indicates that the saved tag is to be used. The *aliasNum* argument has no significance except to give an ID to the relation override specification. If you later realized that you should have used fy92 data, you would again call **setRelationInfo** using the same *aliasNum* value of 1. To override the parameters of a different relation without losing the fy override, use an *aliasNum* of 2 for the second override.

setReportPick

BOOL FAR PASCAL setReportPick (int *hReport*, **char** *cPickFlag*);

hReport	Report handle.
cPickFlag	Report-selection-flag buffer.

Return Value

The **setReportPick** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setReportPick** to replace the current value of the report-selection flag for the report specified by *hReport* to the value specified by *cPickFlag*. If the report-selection flag is set to **R**, the Viewer will prompt the user to select a report from the current report library. If the flag is set to **?**, the Viewer will prompt the user to select a succession of reports from the current report library. The current report library is the library specified explicitly via **setLibrary**, or implicitly via **chooseReport** or **getRuntimeRecord**.

Related Functions

getReportPick, **chooseReport**, **getRuntimeRecord**, **setLibrary**, **getLibrary**

Example

To allow the user to select a report interactively in Viewer for the report whose handle is *hRpt*:

```
setReportPick (hRpt, 'R');
```

setScopeUsage

BOOL FAR PASCAL setScopeUsage (int *hReport*, **char** *cScopeFlag*);

hReport	Report handle.
cScopeFlag	Scope-usage flag.

Return Value

The **setScopeUsage** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setScopeUsage** to set the "scope usage" parameter for the report specified by *hReport* to the value specified by *cScopeFlag*. Valid values for this parameter are:

- **S** (Saved) means to use the scope values saved with the report.
- **E** (Entire) means to ignore any scope values.
- **O** (Override) means to override the saved scope values with the values specified via **setLowScope** and **setHighScope**. (Be careful to use the letter **O** and not the digit zero.)
- Question mark (**?**) allows the user to enter or change scope values. When the scope-usage parameter contains a question mark, the dialog box shown in Figure 3.5 displays. If the window-title parameter (set by **setWinTitle**) is specified, the title bar will contain the that value. If the window-title parameter is blank or has not been set, the title bar will contain the report name.

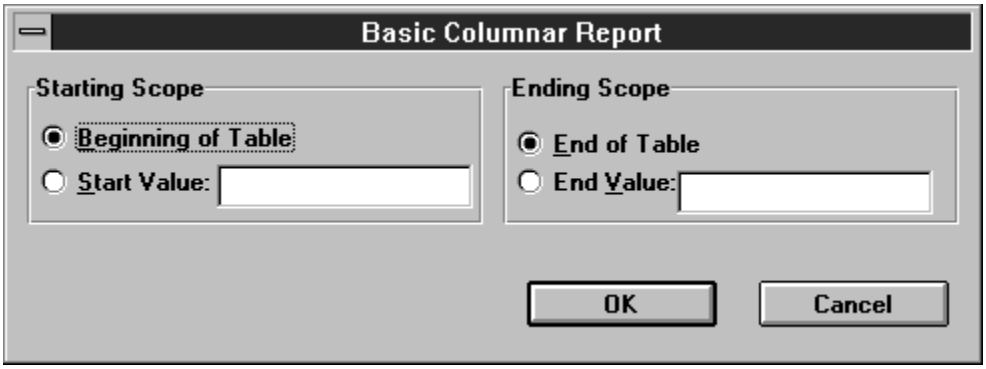

Figure 3.5 High and Low Scope Dialog Box

If the scope-usage parameter is **O** (Override), the low-scope parameter specifies the starting value of the scope and high-scope parameter specifies the ending value of the scope. If the scope-usage parameter is **S**, **E**, or **?**, Viewer ignores the low-scope and high-scope parameters.

Each scope parameter can contain either a record number or an index key value up to 250 characters wide. If no master index was saved with the report (or added using **setMasterIndexInfo**), the Viewer assumes the value is a record number. Otherwise, Viewer assumes the value is a key value in the master index. In this case, the report begins reading the master table at the first record greater than or equal to the low-scope parameter and stops reading the master table after the last record found that is less than or equal to the high-scope parameter.

The range fully includes the end points. In other words, if you enter **A** as the low value and **M** as the high value, Viewer reads the first record in which the value begins with **A** through the last record in which the value begins with **M**. For example, if you have a customer table indexed on last name and you want to print invoices for all customers whose name begins with a letter between **A** and **M**, call **setScopeUsage** with an *lpszScopeFlag* value of **O**, call **setLowScope** with an *lpszScope* value of **A**, and call **setHighScope** with an *lpszScope* value of **M**.

All scope parameters must be character strings. Note that a date scope value must be in the format mm/dd/yy or mm/dd/yyyy. Do not enclose scope values within quotes.

Related Functions

getScopeUsage, **setMasterIndexInfo**, **getMasterIndexInfo**

Example

Assuming we have an index on the NAME field of the master table, to limit the records from the master table to those where the NAME field begins with a letter between **A** and **M**, inclusive, for the report whose handle is *hRpt*:

```
setScopeUsage (hRpt, 'O');    // override saved scope
setMasterIndexInfo (hRpt, (LPSTR)"c:\\data\\name.ndx", 'C', NULL);
setLowScope (hRpt, (LPSTR)"A");
setHighScope (hRpt, (LPSTR)"M");
```

setSortField

BOOL FAR PASCAL setSortField (int *hReport*, **LPSTR** *lpszName*, **int** *sortNum*);

hReport	Report handle.
lpszName	Sort-field-name buffer.
sortNum	Sort-field number.

Return Value

The **setSortField** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setSortField** to replace an existing sort field or add a new one to the report specified by *hReport*. Pass the sort field number to be added or replaced in *sortNum* and its value in *lpszName*. The *lpszName* argument begins with a + or - to indicate ascending or

descending, respectively, followed by the name of the sort field. You must replace all sort fields from sort field 1 through the last sort field you wish to replace. For example, if you only wish to replace sort field 2, you must call **setSortField** twice, once with a *sortNum* of 1 and once with a *sortNum* of 2. To obtain the current sort field parameters, use **getFirstSortField** and **getNextSortField**.

Related Functions

getFirstSortField, **getNextSortField**, **setGroupField**, **getFirstGroupField**, **getNextGroupField**

Example

To replace the second sort field with CITY in ascending order, while leaving the first sort field unchanged, for the report whose handle is *hRpt*:

```
{
   char buf[80];
   getFirstSortField (hRpt, (LPSTR)buf, 80);
   setSortField (hRpt, (LPSTR)buf, 1);
   setSortField (hRpt, (LPSTR)"+CITY", 2);
}
```

setStatusEveryPage

BOOL FAR PASCAL setStatusEveryPage (int *hReport***, BOOL** *bStatus***);**

hReport	Report handle.
bStatus	Status-frequency value.

Return Value

The **setStatusEveryPage** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setStatusEveryPage** to specify a value for the "status every page" parameter for the report specified by *hReport*. This parameter is meaningful only when **execRuntime** is to be called with a value of zero for *bWait*, in which case a status file will be generated. If *bWait* is non-zero, no runtime status file is generated and status is returned to the calling application via **execRuntime**. If *bStatus* is non-zero and a status file is being generated, the file will be updated after each page of the report; otherwise, it will updated only at the end of the report. When *bStatus* is non-zero, you can use the value of RO_PAGES in the status file to restart a report at the point where abnormal termination occurred. See **execRuntime** for more information on restarting reports.

Related Functions

getStatusEveryPage

Example

To specify that runtime status should be written after every page of the report whose handle is *hRpt*:

```
setStatusEveryPage (hRpt, 1);
```

setStatusFileName

BOOL FAR PASCAL setStatusFileName (int *hReport*, **LPSTR** *lpszPath*);

hReport	Report handle.
lpszPath	Status file name

Return Value

The **setStatusFileName** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setStatusFileName** to specify a status file name for the report specified by *hReport*. A status file is created only if you call **execRuntime** with a *bWait* parameter of 0. You can distinguish Viewer status tables by using the **setStatusFileName** to specify the directory in which the file will be created and/or to specify the complete status file name.

To specify the directory in which a status table should be created, specify a full path and name. If you specify a path without a table name, the Viewer executable will create a file named RRUNOUT.DBF in the specified directory. If you specify a filename without a path, the specified file will be created in the current directory.

Example

To cause the Viewer executable to create a status file named C:\TEMP\RUNSTATS.DBF for the report specified by *hRpt*:

```
setStatusFileName (hRpt, "c:\\temp\\runstats.dbf");
```

setSuppressTitle

BOOL FAR PASCAL setSuppressTitle (int *hReport*, **BOOL** *bValue*);

hReport	Report handle.
bValue	Suppress-title flag.

Return Value

The **setSuppressTitle** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setSuppressTitle** to set the "suppress title and summary areas" flag for the report specified by *hReport*. If the value of *bValue* is non-zero, Viewer will not output Title and

Summary areas for reports which contain no records; otherwise Viewer always outputs Title and Summary areas.

Example

To suppress the printing of Title and Summary areas if the report specified by *hRpt* contains no records:

```
setSuppressTitle (hRpt, 1);
```

setTestPattern

BOOL FAR PASCAL setTestPattern (int *hReport*, **BOOL** *bTest*);

hReport	Report handle.
bTest	Test-pattern flag.

Return Value

The **setTestPattern** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setTestPattern** to set the "test pattern" flag for the report specified by *hReport*. If the value of *bTest* is non-zero, Viewer will display a dialog allowing the user to print a test pattern before printing the report. The dialog will contain OK, Cancel, and Print buttons. The user can select OK to print a test pattern as many times as necessary to align forms in the printer, and then select Print to print the report. A test pattern includes only page header, record, and page footer lines.

Related Functions

getTestPattern

Example

To specify that the user should be permitted to print one or more test patterns before printing the report whose handle is *hRpt*:

```
setTestPattern (hRpt, 1);
```

setUserParam

BOOL FAR PASCAL SetUserParam (int *hReport*, **LPSTR** *lpszName*, **LPSTR** *lpszValue*);

hReport	Report handle.
lpszName	Parameter-name buffer.
lpszValue	Parameter-value buffer.

Return Value

The **setUserParam** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setUserParam** to give the value specified by *lpszValue* to the user parameter whose name is specified by *lpszName* for the report specified by *hReport*.

When the Viewer is called directly using a Control File, a user parameter is a control-file field that is not defined by R&R. The value of a user parameter is obtained within a report via the RIPARAM function. When the Viewer is called via the Viewer DLL, the DLL deduces the names of the user parameters by searching all calculated fields for uses of the RIPARAM function. The order in which **getFirstUserParam** and **getNextUserParam** return user parameters is not significant. A given user parameter will only have a current value if **setUserParam** has previously been called for that parameter. All user parameters must be of data type character. You can use conversion functions such as CTOD( ) and VAL( ) to convert to other data types for use in calculations.

You can control some features of the layout and content of reports at runtime by prompting users to enter values for parameters, then passing the values to reports. Typically, you prompt the user for a text string or other data item that is not stored in the database. For example, you might prompt the user for his or her name and use the name in a "Report Author" field in the page footer or title.

Follow these general steps to pass parameters to reports using **setUserParam**.

1. Define calculations in your report using the RIPARAM( ) function.
2. Obtain values for use in the calculations in either of two ways:
   - Create your own menus or prompts within your application.
   - Enter a question mark as the value of the control table field.
3. If your application has obtained values for user parameters, pass the values via calls to **setUserParam**; if you wish Viewer to obtain values for you, call **setUserParam** for each such parameter with a value of questions mark (**?**).

The following sections describe each step in detail.

Define RIPARAM Calculations

In your report, define calculations that obtain user-supplied data via the RIPARAM( ) function. The RIPARAM( ) function takes a user parameter name as its argument and returns the parameter's value as a string.

For example, in a general ledger application, you might define a user parameter named CONAME for the company name, then prompt the user to enter a company name.

To use the company name on the report, create a calculated field in Report Writer whose expression is:

```
RIPARAM("CONAME")
```

You can place the calculated field wherever you want the company name to appear on the report.

Although this example uses an RIPARAM( ) calculated field to provide user input as text in the report, you can use such fields to perform many different functions in a report. For example, you might prompt the user for a value for a DISCOUNT field. In the calculated field on the report, you can convert the user-entered character data to numeric using a calculated field expression such as:

```
ORDERTOT * VAL(RIPARAM("DISCOUNT"))
```

Prompting for User Input

You can get user input in two ways:

- Supply a menu or prompt in your application that leads the user to supply a value. Pass this value to the Viewer DLL via **setUserParam**.
- Enter a question mark (**?**) value for any user-defined field. Whenever a user-defined field contains a question mark, the user will be prompted to enter a value.

Using the Question Mark Field Value

The simplest way to get user input for reports is to use a question mark (**?**) as the value for a user parameter. Optionally, the value can also include the text you want to appear as a prompt. For example, if you want to prompt the user for his or her name, you might define an AUTHOR user parameter and give it the value "?Enter your name:". At runtime, the user will see the dialog shown in Figure 3.6.

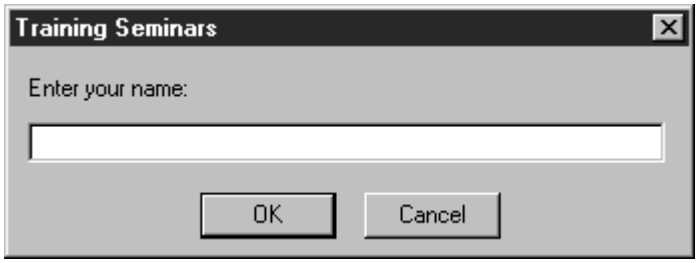

Figure 3.6 Viewer Dialog Box with Prompt

The size and shape of this dialog box is the same for all user-defined fields. The title bar contains the title set with **setWinTitle**. If **setWinTitle** has not been called, the Viewer uses the report name. If the user selects the Cancel button, the report will not run and the Viewer will write the "Canceled" message to the status file.

If your control table field contains a question mark only and no text string, the Viewer displays the dialog box shown in Figure 3.6 with the prompt "Enter value for (USER PARAMETER)", as in "Enter value for AUTHOR".

Passing Parameter Values to the Viewer DLL

After obtaining values for user parameters, the final step is to pass those values to the *Viewer* DLL so they become available for use in RIPARAM() calculations. Use **setUserParam** to specify values for user parameters.

Related Functions

getFirstUserParam, **getNextUserParam**

Example

To specify, for the report whose handle is *hRpt*, a value of R. T. Firefly for the user parameter named AUTHOR:

```
setUserParam (hRpt, (LPSTR)"AUTHOR", (LPSTR)"R. T. Firefly");
```

setWinBorderStyle

BOOL FAR PASCAL setWinBorderStyle (int *hReport***, int** *style***);**

hReport	Report handle.
style	Preview window border style.

Return Value

The **setWinBorderStyle** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setWinBorderStyle** to specify the type of border for the preview window for the report specified by *hReport*. The valid values for *style* are:

- If *style* is 1, the preview window will be fixed size with a standard border.
- If *style* is 2, the user will be able to change the size of the preview window.

Related Functions

setWinControlBox, **setWinHeight**, **setWinLeft**, **setWinMaxButton**, **setWinMinButton**, **setWinTitle**, **setWinTop**, **setWinWidth**

Example

To specify that the preview window for the report whose handle is *hRpt* should have a single-line border and a fixed size of 400 pixels wide and 300 pixels high:

```
setWinBorderStyle (hRpt, 1);
setWinWidth (hRpt, 400);
setWinHeight (hRpt, 300);
```

setWinControlBox

BOOL FAR PASCAL setWinControlBox (int *hReport***, BOOL** *bControlBox***);**

hReport	Report handle.

bControlBox Preview window control box flag.

Return Value

The **setWinControlBox** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setWinControlBox** to specify whether the preview window is to have a control box in the upper-left corner for the report specified by *hReport*. If *bControlBox* is non-zero, the preview window will have a control box.

Related Functions

setWinBorderStyle, **setWinHeight**, **setWinLeft**, **setWinMaxButton**, **setWinMinButton**, **setWinTitle**, **setWinTop**, **setWinWidth**

Example

To specify that the preview window for the report whose handle is *hRpt* should have a control box, and a maximize button, but no minimize button:

```
setWinControlBox (hRpt, 1);
setWinMaxButton (hRpt, 1);
setWinMinButton (hRpt, 0);
```

setWinHeight

BOOL FAR PASCAL setWinHeight (int *hReport***, int** *height***);**

hReport Report handle.

height Preview window height.

Return Value

The **setWinHeight** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setWinHeight** to specify the height in pixels of the preview window for the report specified by *hReport*.

Related Functions

setWinBorderStyle, **setWinControlBox**, **setWinLeft**, **setWinMaxButton**, **setWinMinButton**, **setWinTitle**, **setWinTop**, **setWinWidth**

Example

To specify that the preview window for the report whose handle is *hRpt* should have a single-line border and a fixed size of 400 pixels wide and 300 pixels high:

```
setWinBorderStyle (hRpt, 1);
setWinWidth (hRpt, 400);
setWinHeight (hRpt, 300);
```

setWinLeft

BOOL FAR PASCAL setWinLeft (int *hReport***, int** *left***);**

hReport	Report handle.
left	Preview window left-edge position

Return Value

The **setWinLeft** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setWinLeft** to specify the position of the left edge of the preview window for the report specified by *hReport*. *left* specifies how far, in pixels, from the left edge of the screen the left edge of the preview window is to be.

Related Functions

setWinBorderStyle, **setWinControlBox**, **setWinHeight**, **setWinMaxButton**, **setWinMinButton**, **setWinTitle**, **setWinTop**, **setWinWidth**

Example

To specify that the preview window for the report whose handle is *hRpt* should begin 50 pixels down and 40 pixels to the right of the upper-left corner of the screen:

```
setWinTop (hRpt, 50);
setWinLeft (hRpt, 40);
```

setWinMaxButton

BOOL FAR PASCAL setWinMaxButton (int *hReport***, BOOL** *bMaxButton***);**

hReport	Report handle.
bMaxButton	Preview window maximize-button flag.

Return Value

The **setWinMaxButton** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setWinMaxButton** to specify whether the preview window is to have a maximize button. If *bMaxButton* is non-zero the preview window will have a maximize button.

Related Functions

setWinBorderStyle, **setWinControlBox**, **setWinHeight**, **setWinLeft**, **setWinMinButton**, **setWinTitle**, **setWinTop**, **setWinWidth**

Example

To specify that the preview window for the report whose handle is *hRpt* should have a control box, and a maximize button, but no minimize button:

```
setWinControlBox (hRpt, 1);
setWinMaxButton (hRpt, 1);
setWinMinButton (hRpt, 0);
```

setWinMinButton

BOOL FAR PASCAL setWinMinButton (int *hReport*, **BOOL** *bMinButton*);

hReport	Report handle.
bMinButton	Preview window minimize-button flag.

Return Value

The **setWinMinButton** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setWinMinButton** to specify whether the preview window is to have a minimize button. If *bMinButton* is non-zero the preview window will have a minimize button.

Related Functions

setWinBorderStyle, **setWinControlBox**, **setWinHeight**, **setWinLeft**, **setWinMaxButton**, **setWinTitle**, **setWinTop**, **setWinWidth**

Example

To specify that the preview window for the report whose handle is *hRpt* should have a control box, and a maximize button, but no minimize button:

```
setWinControlBox (hRpt, 1);
setWinMaxButton (hRpt, 1);
setWinMinButton (hRpt, 0);
```

setWinTitle

BOOL FAR PASCAL setWinTitle (int *hReport*, **LPSTR** *lpszTitle*);

hReport	Report handle.
lpszTitle	Report title.

Return Value

The **setWinTitle** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setWinTitle** to set the value of the “report title” parameter for the report specified by *hReport* to the text specified by *lpszTitle*.

The report title is displayed in the following places:

- The title bar of the Preview window
- The Print Status window (if **setDisplayStatus** is called with a non-zero *bStatus* value.)
- Below the Viewer icon (if **setDisplayStatus** is called with a *bStatus* value of zero.)
- The title bar of the dialog boxes that display if **setPrinter** or **setPrinterPort** is called with an *lpszPrinter* value of question mark, or if **setScopeUsage** is called with an *lpszScopeFlag* value of question mark.

If this field is blank, the Viewer will use the name of the report as the window title.

Related Functions

getWinTitle, setStatus, setPrinter, setScopeUsage

Example

To specify that the preview window for the report whose handle is *hRpt* should have a title of "on the desktop of Rufus T. Firefly":

```
setWinTitle (hRpt, (LPSTR)"on the desktop of Rufus T. Firefly");
```

setWinTop

BOOL FAR PASCAL setWinTop (int *hReport***, int** *top***);**

hReport	Report handle.
top	Preview window top-edge position.

Return Value

The **setWinTop** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setWinTop** to specify the position of the top edge of the preview window for the report specified by *hReport*. *top* specifies how far, in pixels, from the top edge of the screen the top edge of the preview window is to be.

Related Functions

setWinBorderStyle, **setWinControlBox**, **setWinHeight**, **setWinLeft**, **setWinMaxButton**, **setWinMinButton**, **setWinTitle**, **setWinWidth**

Example

To specify that the preview window for the report whose handle is *hRpt* should begin 50 pixels down and 40 pixels to the right of the upper-left corner of the screen:

```
setWinTop (hRpt, 50);
setWinLeft (hRpt, 40);
```

setWinWidth

BOOL FAR PASCAL setWinWidth (int *hReport*, **int** *width*);

hReport	Report handle.
width	Preview window width.

Return Value

The **setWinWidth** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setWinWidth** to specify the width, in pixels, of the preview window for the report specified by *hReport*.

Related Functions

setWinBorderStyle, **setWinControlBox**, **setWinHeight**, **setWinLeft**, **setWinMaxButton**, **setWinMinButton**, **setWinTitle**, **setWinTop**

Example

To specify that the preview window for the report whose handle is *hRpt* should have a single-line border and a fixed size of 400 pixels wide and 300 pixels high:

```
setWinBorderStyle (hRpt, 1);
setWinWidth (hRpt, 400);
setWinHeight (hRpt, 300);
```

setWriteAllow

BOOL FAR PASCAL setWriteAllow (int *hReport*, **BOOL** *bAllow*);

hReport	Report handle.
bAllow	Allow-write flag.

Return Value

The **setWriteAllow** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setWriteAllow** to set the "allow write" flag for the report specified by *hReport*. If the value of *bAllow* is non-zero, Viewer will open database and index files in a mode that allows them to be modified. The allow-write flag specified via **setWriteAllow** overrides the flag value specified in RRW.INI. Even if the allow-write flag is set to false, other R&R users will be able to report on files you are using.

For more information about file write access settings, see the explanation of the Allow Other Users to Update Database Tables setting in Chapter 5, "Setting Defaults," in *Using*

R&R. Note that this switch controls R&R's behavior only when accessing shared data; your database or network software may impose other file access restrictions.

Related Functions

none

Example

To specify that Viewer should allow other users to modify databases and indexes it is using to generate the report specified by *hRpt*:

```
setWriteAllow (hRpt, 1);
```

setXbaseEditor

BOOL FAR PASCAL setXbaseEditor (int *hReport*, **BOOL** *bXbase*);

hReport	Report handle.
bXbase	Xbase-memos flag.

Return Value

The **setXbaseEditor** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **setXbaseEditor** to set the "Xbase memos" flag for the report specified by *hReport*. If the value of *bXbase* is non-zero, Viewer will assume that all database memos in use by this report were created with an Xbase memo editor.

Related Functions

none

Example

To specify that the database memos for the report whose handle is *hRpt* were create with an Xbase memo editor:

```
setXbaseEditor (hRpt, 1);
```

writeRuntimeRecord

BOOL FAR PASCAL writeRuntimeRecord (int *hReport*, **LPSTR** *lpszControlFile*);

hReport	Report handle.
lpszControlFile	Job-control-filename buffer.

Return Value

The **writeRuntimeRecord** function returns zero if an error occurs. To obtain more information about the error use **getErrorInfo**.

Description

Use **writeRuntimeRecord** to save all parameters for the report specified by *hReport* to the ASCII Viewer Control File specified by *lpszControlFile*. If *lpszControlFile* is the NULL pointer or contains the null string, **writeRuntimeRecord** will overwrite the Viewer Control File read via **getRuntimeRecord**. If the *hReport* was not returned from **getRuntimeRecord**, *lpszControlFile* must contain a filename.

Related Functions

getRuntimeRecord, **execRuntime**

Example

To read an existing ASCII Viewer Control File, modify some parameters and then save the results in the same file:

```
{
   int hRpt;
   if (hRpt = getRuntimeRecord ((LPSTR)"App Name",
      (LPSTR)"c:\\rrdata\\runrecd"))
   {
      setScopeUsage (hRpt, 'E');
      setFilterUsage (hRpt, 'E');
      writeRuntimeRecord (hRpt, NULL);
   }
}
```

Chapter 4
Using the Custom Control

Introduction

As noted in Chapter 1, the Viewer OCX (also referred to as a *custom control*) provides one of three methods for running reports using the Viewer. The other methods are explained in Chapter 2, "Using the Viewer Executable," and Chapter 3, "Accessing the Viewer DLL."

The Viewer OCX allows you to incorporate database reporting into your applications and provides the following advantages:

- ❑ Access to the powerful reporting capabilities of R&R;
- ❑ Extensive control over report characteristics by means of more than 50 properties that can be set at design time or run time;
- ❑ Design-time support in the form of dialog boxes that allow you to point and click to override settings in your report, such as tables, sorting and grouping, user parameters, and destination.

The Viewer OCX is explained in the following sections:

- Installation
- Determining Report Status
- Using RRW.INI for Default Information
- Using the Custom Control
- Custom Control Properties

Installation

The custom control is installed if you select a Setup Type of "Typical" or if you select "Custom" and specify Viewer as one of the options. Setup also copies sample VB (Visual Basic) and C projects into approprate subdirectories of the program directory.

To add the R&R custom control to an existing VB project, select Tools ⇒ Custom Controls to add the file RRW32.OCX, which Setup installs into your Windows System directory. For information about other files you will need to distribute when you use the R&R custom control in your applications, see Chapter 7, "Distributing Reports."

Determining Report Status

When you use the custom control to print a report, you may want to know the status of the report, such as whether the report printed successfully or, if not, which error occurred while printing. How the control returns this status to you depends on which method of report printing you use.

The custom control supports two methods of report printing, *synchronous* and *asynchronous*. Synchronous printing means that the report will complete printing before the next line of procedure code is executed. Asynchronous printing means that the report will be printed while the remaining lines of procedure code are executing. Each method has its own way of returning status information to you.

You print a report synchronously by setting the Action property to 1 (e.g., RRReport1.Action = 1). When you print a report synchronously, the status of the report is returned in the following properties:

- LastErrorCode, which will contain the type of error that occurred, or 0 for no error;
- LastErrorString, which will contain a text message describing the error, if any;
- LastErrorPage, which will contain the page number of the last page printed.

You print a report asynchronously by setting the Action property to 2 (e.g., RRReport1.Action = 2). When you print a report asynchronously, the status of the report is not returned in a property, but is written into the Viewer status file, which is a text file called RRWRUN.OUT in the current working directory. See the section entitled **Understanding the Viewer Status File** in Chapter 2 for a description of the contents of this file.

Using RRW.INI for Default Information

If RRW.INI, the R&R configuration file, is in the Windows directory, the Viewer will use the default settings specified in that file. However, custom control properties take precedence; any setting you specify using a custom control property will always override the corresponding RRW.INI setting (see Figure 2.2 in Chapter 2 for a list of RRW.INI settings that the Viewer will use).

If you distribute reports to other users, you can customize RRW.INI for each user and distribute it with the other Viewer files. However, the custom control properties provide a more reliable way to control or override the defaults saved with a report.

Using the Custom Control

You use the Viewer OCX just like any other OCX control. For example, in Visual Basic simply click on the R&R tool in the VB Toolbox. Then move the mouse pointer over your form, press the left mouse button down, and drag the mouse. When you release the mouse button, the custom control will be placed onto the form.

You can change the value of a control property in either of two ways:

- Enter or select values on the appropriate property pages of the Report Control Properties dialog;
- Directly enter or select values for each property on the Properties list.

Note that the R&R custom control will be visible at design time only. At run time, it will not be displayed, but will be “at your service” to print your reports to a printer; to a preview window; or to a text, database, or spreadsheet file. It does this by invoking the Viewer executable (described in Chapter 2) through the Viewer DLL (described in Chapter 3). As you will see, the custom control properties correspond closely to the control parameters used by the Viewer. (See Appendix A, “Runtime Equivalencies.”)

To print or display a report in your program, you must set at least two properties:

1. Set the ReportName property to the name of the report you want to print; alternatively, set the ReportPick property either to 1 (One) to prompt the user to select a report or 2 (Many) to prompt the user to select several reports in succession.
2. Set the Action property to 1 to trigger execution of the report.

The ReportName and ReportPick properties can be set at design time or at run time. The Action property must be set at run time in your procedure code (e.g., RRReport1.Action = 1). You can also set many other properties (Destination, SortFields, etc.) to override the values saved in the report.

Changing Values Using the Properties List

If you use the Property list to set or change values, the data type of the property determines how you change its value:

- To change an integer property, enter a value in the settings box.
- To change an enumerated property, either type a number into the settings box or select a value from the drop-down list that appears when you click on the down arrow next to the settings box. You can also double-click on an enumerated property to cycle through the list of values.
- You change string properties by typing a string in the settings box. You can change many string properties by means of dialog boxes that appear when you click on the ellipsis (...) to the right of the settings box, or when you double-click on the property.
- You set some properties by selecting or entering either True (to turn the setting on) or False (to turn the setting off).

Changing Values Using the Control Properties Dialog

The Control Properties dialog (see Figure 4.1) consists of 11 property pages for controlling the more than 50 custom control properties.

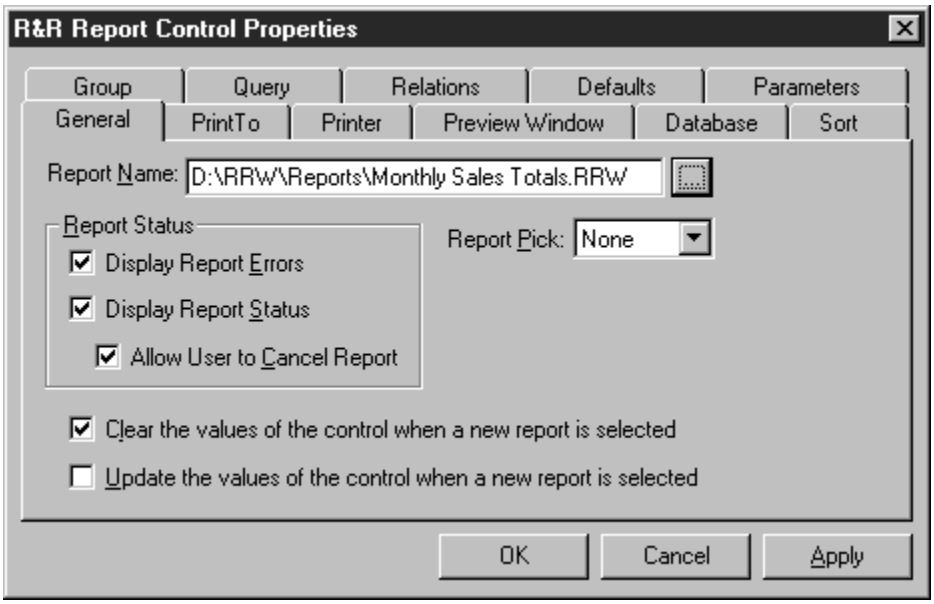

Figure 4.1 Report Control Properties Dialog Box

To set or change values on a property page, do the following:

1. Select the appropriate tab to open the property page.
2. Change values by clicking buttons/checkboxes, entering text into text boxes, and selecting ellipses buttons (where available) to choose from browse dialogs.
3. Select Apply (or click another tab) to save your changes.
4. Repeat Steps 1 – 3 for each property page as necessary.
5. When you are finished setting or changing values, select OK to close the Control Properties dialog box.

The following sections briefly explain the settings on each of the property pages. See the Custom Control Properties section for additional information about individual properties, including examples and descriptions of how to set values in your procedure code.

General Property Page

Settings on the General property page control report selection and status checking.

Setting	*Purpose*
Report Name	Sets the **ReportName** property to specify the report to be run. Either enter the path and name of the report or select the ellipses button to display the Open dialog to select a report file.
Clear ...	Sets the **ResetProperties** property to specify whether settings are reset to their default values when a new report is selected.
Update ...	Sets the **UpdateControl** property to specify whether settings should be updated with the properties of the saved report when a new report is selected.
Display Report Errors	Sets the **DisplayError** property to either True (display errors) or False (don't display errors.
Display Report Status	Sets the **DisplayStatus** property to either True or False to control whether a Print Status window is displayed when a report is printed.
Allow User ...	Sets the **NoEscape** property to control whether a Cancel button is available on the Print Status window (when Display Report Status is set to True).
Report Pick	Sets the **ReportPick** property to enable the user to choose a report (when set to 1) or a series of reports (when set to 2) at Viewer.

Print To Property Page

Settings on the Print To property page control or override the print and/or export destination of the Viewer report.

Setting	*Purpose*
Report Destination	Sets the **Destination** property to specify the report destination: Saved Destination, Prompt User, Preview Window, Printer, or Export. If set to Export, a drop-down list of export types is available (Text, DBF, WKS, RTF, HTML, ActiveX control, Text Data, or Word Merge file)
Excel Export Destination	Sets the **ExportDestination** property to specify the destination (Preview Window, Printer, or File) for an export type of Excel PivotTable or Excel Chart. If File is selected, enter the output name in the File Name box.
File Name	Sets the **PrintFileName** property to specify the output file for any export to file.

Printer Property Page

Settings on the Printer property page control or override the destination printer for the Viewer report.

Setting	*Purpose*
Printer Destination	Sets the **Printer** property to specify whether to use the saved printer, override the saved printer, or prompt the user at Viewer for printer selection.
Print a Test Pattern	Sets the **TestPattern** property to specify whether a test pattern will be printed for purposes of checking report layout before printing.
Start Page	Sets the **StartPage** property to specify the page at which to begin printing.
End Page	Sets the **EndPage** property to specify the page at which to end printing.
Number of Copies	Sets the **CopiesToPrinter** property to specify how many report copies to print.

Preview Window Property Page

Settings on the Preview Window property page control the location, dimensions, and title of the Viewer preview window.

Setting	*Purpose*
Left	Sets the **WindowLeft** property to specify the starting point for the left edge of the preview window.
Height	Sets the **WindowHeight** property to specify the height (in pixels) of the preview window.
Top	Sets the **WindowTop** property to specify the starting point for the top edge of the preview window.
Width	Sets the **WindowWidth** property to specify the width (in pixels) of the preview window.
Minimize Button	Sets the **WindowMinButton** property to control whether the preview window will have a minimize control in the caption bar.
Maximize Button	Sets the **WindowMaxButton** property to control whether the preview window will have a maximize control in the caption bar.
Border	Sets the **WindowBorderStyle** property to control whether the preview window will be fixed or variable size.
Title	Sets the **WindowTitle** property to specify the text that will appear in the title bar of the preview window, the Print Status window (if Display Report Status is set to True), and in the title bars of the dialog that displays when the Printer, Port, Destination, or Scope value is a question mark.

Database Property Page

The Database property page settings control several report database characteristics, including master table and index selection and scope settings.

Setting	*Purpose*
Master Table	Sets the **MasterTable** property to override the master table saved with the report.
Master Index	Sets the **MasterIndex** property to specify or override the master index selection.
Type	Sets the data type argument to the MasterIndex property.
Tag	Sets the tag argument to the MasterIndex property (for multiple-field index files).
Memo Filename	Sets the **MemoFileName** property to specify the name and location of the text memo file to be used by the Viewer report.
Saved Scope	Sets the **Scope** property to control use of the saved scope values, override of the scope, or prompting of the user to supply scope values.
Low Scope	If Scope is set to 2 (Override), this sets the **LowScope** property to specify the starting scope. If Scope is set to 2 (Override), this sets the **HighScope** property to specify the ending scope.

Sort Property Page

The Sort property page specifies or overrides the field(s) to be used for sorting of report data. Selecting one or more sort fields in the numbered field-selection slots sets the **SortFieldsString** property accordingly.

To display a drop-down list of available fields, click the arrow at the right of the selection slot.

Group Property Page

The Group property page specifies or overrides the field(s) to be used for grouping of report data. Selecting one or more group fields in the numbered field-selection slots sets the **GroupFieldsString** property accordingly.

To display a drop-down list of available fields, click the arrow at the right of the selection slot.

Query Property Page

The Query property page specifies or overrides the query to be used to filter the report data.

Setting	*Purpose*
Saved Query	Sets the **Query** property to specify using the saved query, ignoring the saved query, overriding the saved query, or prompting the user.
Query Override	If Saved Query is set to Override, this sets the **Filter** property to specify the query to be used.

Relations Property Page

The Relations property page specifies or overrides the related tables for the Viewer report.

Setting	*Purpose*
Relate Tables	Sets the **RelatedTablesString** property to specify or override table relations.

Defaults Property Page

The Defaults property page specifies or overrides defaults for data, image, and report directories and for index file extension.

Setting	*Purpose*
Data	Sets the **DataDirectory** property to specify the default directory for data files used by a report.
Image	Sets the **ImageDirectory** property to specify default location for image files.
Report	Sets the **ReportDirectory** property to specify default location for report or library files.
Memo Editor	Sets the **XbaseEditor** property to specify which memo editor was used to prepare database memos for use in reports.
Index Extension	Sets the **IndexExtension** property to specify a default index file extension.
Allow other users...	Sets the **WriteAllow** property to specify whether database users can modify tables used by reports.

Parameters Property Page

The **ParametersString** property page specifies user parameter values to be used with the Viewer report.

Setting	*Purpose*
User Parameters	Name(s) of the user-defined parameter(s) in the report, as specified using the RIPARAM( ) function.
Value	Corresponding value to be assigned to each user parameter at Viewer.

Custom Control Properties

The following properties are available in the R&R custom control:

(About)

Description

Double-click About to display the version of the R&R custom control.

Availability

Design time only

Action

Description

Action is a property that triggers the print, display, or export of the report.

Usage

[form.]*ControlName*.Action [=Action%]

Example

```
RRReport1.Action = 1
```

« Prints, displays, or exports the report, depending on the Destination property, and does not return until the report is completed. »

Comments

Set the Action property to 1 or 2 in your procedure code to print, display, or export the report in response to a user event. In most cases, it will be more convenient to set this property to 1.

If set to 1, the action is synchronous, which means that the next line of Visual Basic procedure code will not execute until the report is completed. The status of the report will be returned in the LastErrorCode, LastErrorString, and LastErrorPage properties.

If set to 2, the action is asynchronous, so that the report may still be running when the next line of Visual Basic procedure code is executed. When the report completes, its status is written into the status file.

Data Type

Integer

Availability

Write-only at run time

CopiesToPrinter

Description

Specifies the number of copies to be printed if you are printing to a printer (if the Destination property is set to 1).

Usage

[form.]*ControlName*.CopiesToPrinter[= NumCopies%]

Example

```
RRReport1.CopiesToPrinter = 3
```

« Prints three copies of the report. »

Comments

This property is optional. The number must be between 0 and 999, inclusive. If you leave this property blank or enter 0, the Viewer prints the number of copies saved with the report.

Data Type

Integer

Availability

Design time; Run time

DataDirectory

Description

Specifies the default directory where the Viewer will look for tables, indexes, and text memo files to be used when the report is printed.

Usage

[form.]*ControlName*.DataDirectory[= DirectoryName$]

Example

```
RRReport1.DataDirectory = "c:\mis\data"
```

« Looks for data files in a directory called “c:\mis\data.” »

Comments

If the tables, indexes, and text memo files used in the report are not in the saved directories, then the Viewer will look in this directory for these files.

Data Type

String

Availability

Run time

Destination

Description

Specifies the destination to which your report is to be printed or exported (Preview, Printer, Text, DBF, WKS, RTF, text data, Word merge, Excel Chart, Excel PivotTable, ActiveX control, or HTML).

Usage

[form.]*ControlName*.Destination[= Destination%]

Example

```
RRReport1.Destination = 1
```

« Sends the report to a preview window. »

Comments

This property is optional. Set it to 0 (the default) to print to the printer saved with the report (or to the printer specified in the Printer property). This property can contain one of the following values:

0 – Saved (uses the printer saved with the report);
1 – Window (sends the report to a preview window);
2 – Printer (sends the report to a printer);
3 – Text File (exports the report to an ASCII text file);
4 – DBF File (exports the report to a DBF database file);
5 – WKS File (exports the report to a WKS spreadsheet file);

6 – Prompt user (asks user for destination);
7 – RTF File (exports to a Rich Text Format file);
8 – Text Data File (exports to a Text Data file);
9 – Word Merge file (exports to a Word Merge file).
10 – Excel Chart
11 – Excel PivotTable
12 – ActiveX Viewer file
13 – HTML export

If you specify 3 (Text File), 4 (DBF File) 5 (WKS File), 7 (RTF File), 8 (Text Data file), 9 (Word Merge file), 12 (ActiveX), or 13 (HTML), you can also set the PrintFileName property to provide the name of the destination file to override the saved destination file name.

If you select Window, the report will be sent to the display, allowing the user to preview the report before printing it. After previewing the report, the user can select the Print button in the Preview window to send the report to the printer saved with the report or specified in the Printer property. Note that if the value of Destination is 1 and the PrintFileName property has been set, the report will be output to the file specified in PrintFileName when the user selects Print in Preview.

Setting this property to 6 allows the user to select the print destination at run time. The user will see the dialog box shown in Figure 4.2. If the WindowTitle property is set, the title bar will contain the WindowTitle value. If WindowTitle is empty, the title bar will contain the report name.

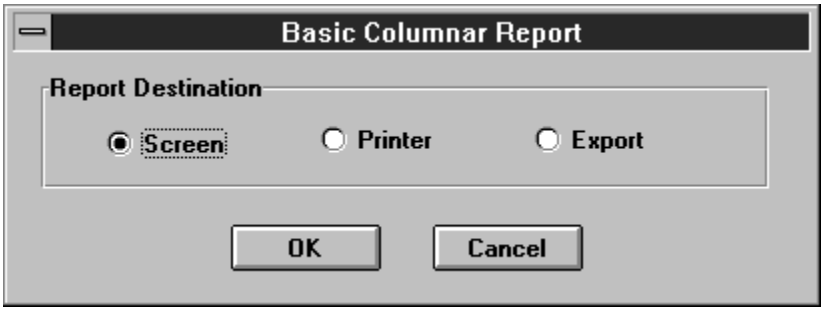

Figure 4.2 Print Destination Dialog Box

The user can select Screen to preview the report, Printer to print it, or Export to display the Export dialog. If the user selects Cancel, the

report will not run, and the "Canceled" status message will be returned in the LastErrorString property or the Viewer status file.

Data Type

Integer (Enumerated)

Availability

Design time; Run time

DisplayError

Description

Specifies whether errors are to be displayed when a report is printed.

Usage

[form.]*ControlName*.DisplayError [= {True|False}]

Example

```
RRReport1.DisplayError = True
```

« Specifies that any errors that occur when a report is printed will be displayed. »

Comments

This property is optional. If DisplayError is True, Viewer error messages are displayed in addition to being returned in the LastErrorString property. In this case, the Viewer stops processing a report when it encounters an error and displays an error message dialog. The user must then select OK to acknowledge the error and resume processing.

If DisplayError is False, Viewer error messages are not displayed, but are returned in the LastErrorString property or the Viewer status table.

Data Type

Integer (Boolean)

Availability

Design time; Run time

DisplayStatus

Description

Specifies whether or not status information is to be displayed when a report is printed.

Usage

[form.]*ControlName*.DisplayStatus [= {True|False}]

Example

```
RRReport1.DisplayStatus = False
```

« Specifies that status information will not be displayed when a report is printed. »

Comments

The DisplayStatus property enables you to specify whether the Viewer should display a Print Status window while it is generating a report. If the property is set to True, the Viewer will display a Status window. If NoEscape is set to False, the Status window will contain a Cancel choice that allows the user to terminate a report in progress.

If DisplayStatus is set to False, the Viewer will not display a Status window but will instead display as an icon while it is running.

Data Type

Integer (Boolean)

Availability

Design time; Run time

EndPage

Description

Specifies at which page of the report to end printing.

Usage

[form.]*ControlName*.EndPage[= Page%]

Example

```
RRReport1.EndPage = 20
```

« Specifies that the report should end printing at the completion of page 20. »

Comments

This property is optional. The StartPage and EndPage properties allow you to override the starting and ending page numbers saved with the report. The default value for these properties is zero.

To specify page numbers, include a StartPage value, an EndPage value, or both. If you specify both, EndPage must be equal to or greater than StartPage. For example, users can restart a canceled report where it was interrupted by specifying the starting page number as the StartPage value and 999999999 as the EndPage value. To reprint one or more consecutive pages, specify the page numbers in the StartPage and EndPage properties. To print just one page, specify the same value for both.

Data Type

Integer

Availability

Design time; Run time

ExportDestination

Description

Specifies the destination (display, file, or printer) when exporting to an Excel PivotTable or Chart.

Usage

[form.]*ControlName*.ExportDestination[= Destination%]

Example

```
RRReport1.ExportDestination = 2
```

« Sends the PivotTable or Chart export to a file. »

Comments

Set this property to one of the following values if you are exporting to an Excel PivotTable or Chart..

0 – Window;

1 – Printer;

2 – File.

If you set the value to 2 (File), use PrintFileName to specify a file name for the export.

Data Type

Integer (Enumerated)

Availability

Design time; Run time

Filter

Description

Specifies a query to select records to be used when printing the report.

Usage

[form.]*ControlName*.Filter[=Filter$]

Example

```
RRReport1.Filter = "Year > 1996"
```

Comments

The optional Filter property specifies a logical expression that will override the query saved with a report, if any, when the value in Query is 2.

The syntax of the Filter expression is identical to that of a calculated field expression that returns a logical value. The Filter expression can be up to 1024 characters long. When an expression is specified, the Viewer selects all records where the value of the Filter expression is true. The expression can refer to any data or calculated fields that are available in the report.

For example, if you enter the expression **CITY="Dallas"**, the Viewer will select all records where the value of this expression is true, in

other words all records where the value in the CITY field is Dallas. If the city name were in a character field named NOTE, the expression **NOTE="*Dallas*"** would select all records in which the NOTE field contained the word "Dallas".

Entering the expression **PASTDUE=T** tells the Viewer to select all records where the value in the PASTDUE field is true. Entering **AMOUNT>=200** will select all records where the value in the AMOUNT field is equal to or greater than 200.

Entering the following expression will select all records where the date in the INVDATE field of the RRORDERS table is January 31, 1996:

```
RRORDERS->INVDATE={01/31/96}
```

Compound expressions can be entered by using parentheses. For example, the following expression selects all records where the value in the CITY field is either Dallas or Houston and where the value in the SALES field is greater than 50,000:

```
(CITY="Dallas" or CITY="Houston") and SALES>50000
```

Note that the value of Query *must* be 2 in order for the Filter override to take effect. If you omit Query, the Filter value will be ignored and the report will be run using the saved query (if any).

When setting this property at run time, make sure that you enclose your query expression in double quotes. If your query expression contains internal quotes, such as:

LNAME = "Jones"

make sure to change all of the internal double quotes to single quotes and then put double quotes around the entire query expression, such as:

"LNAME = ‘Jones’"

At design time, you can change this property array in two ways:

- Double-click this property to display the Query property page (see Figure 4.3), where you can specify a query to override the one saved with the report.

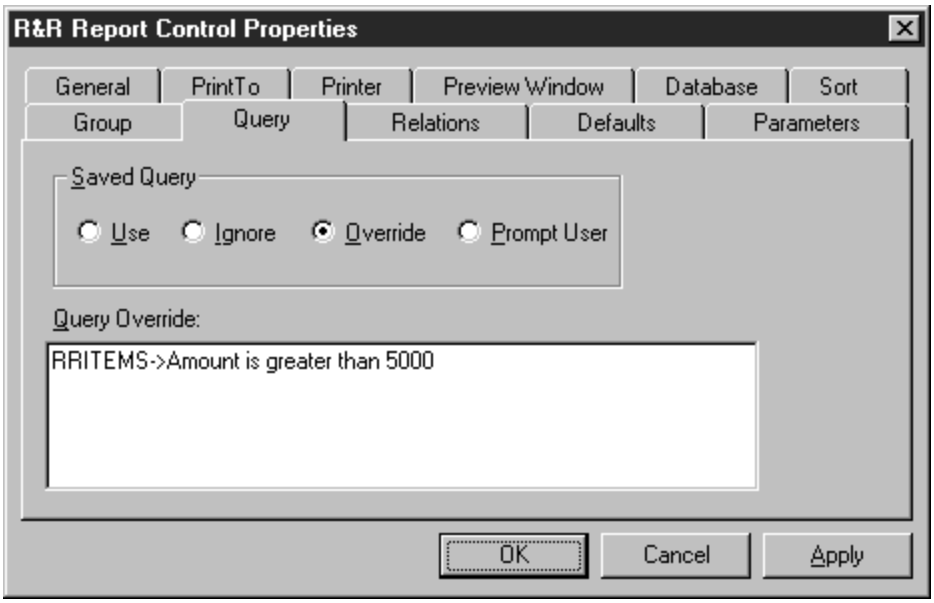

Figure 4.3 Query Property Page

- Simply enter the query expression into the settings box.

Data Type

String

Availability

Design time; Run time

GroupFields

Description

Specifies the field(s) to be used to group the data in your report.

Usage

[form.]*ControlName*.GroupFields(ArrayIndex)[= "+|-GroupField"]

Example

```
RRReport1.GroupFields(0) = "Division"
```

« Use “Division” as the first group field. »

Comments

Group fields can be database fields, calculated fields, or total fields.

When setting this property at run time, use a separate line of code to specify each group field. The first group field you specify must be assigned array index 0, the second group field must be assigned array index 1, etc. The index values you assign must be continuous; no gaps are allowed (0,1,2 would be correct, but 0,1,3 would be wrong).

Data Type

Array of strings

Availability

Run time

GroupFieldsString

Description

Specifies the field(s) to be used to group the data in your report.

Usage

[form.]*ControlName*.GroupFieldsString[= "+|-GroupField1;+|-GroupField2"]

Example

```
RRReport1.GroupFieldsString = "Division"
```

« Use “Division” as the first group field. »

Comments

Group fields can be database fields, calculated fields, or total fields.

At design time, you can change this property array in two ways:

- Double-click this property to display the Group property page, which lists group fields in the report. Clicking on the down arrow next to each group field will drop down a list of all fields used in the report from which you can select.
- Enter group field names separated by semicolons. To override some group fields, but not all of them, you must use a semicolon as a place-holder. For example, to change the first and third group field, you would enter “Division;;Region”.

Data Type

String

Availability

Design time

HighScope

Description

Specifies the high scope of the records to be used when the report is printed.

Usage

[form.]*ControlName*.HighScope[= Scope$]

Example

```
RRReport1.HighScope = "Smith"
```

« Specifies that the scope of records to be used should stop just after "Smith" when the report is printed. »

Comments

Set this property if you want to override the high scope value saved in the report. Be sure to set the Scope property to 2 to use the override. See **Scope** for a complete description of this feature.

Data Type

String

Availability

Design time; Run time

ImageDirectory

Description

Specifies the default directory where the Viewer will look for image files used in the report.

Usage

[form.]*ControlName*.ImageDirectory[= DirectoryName$]

Example

```
RRReport1.ImageDirectory = "c:\mis\images"
```

« Looks for image files in a directory called “c:\mis\images.” »

Comments

The Viewer will look for image files in this directory when they are not in the saved directory. The directory you specify with this property overrides any default image directory specified in the RRW.INI file.

Data Type

String

Availability

Design time; Run time

IndexExtension

Description

Specifies the default index file extension.

Usage

[form.]*ControlName*.IndexExtension[=Extension%]

Example

```
RRReport1.IndexExtension=5
```

« Uses “ntx” as the default index file extension. »

Comments

This property is optional; it can contain one of the following values:

0 – None	4 – NDX (dBASE III)
1 – CDX (FoxPro)	5 – NTX (Clipper)
2 – IDX (FoxBASE+)	6 – NSX (SuccessWare)
3 – MDX (dBASE IV)	7 – WDX (WordTech)

The default index extension is used to locate indexes that are specified without extensions or that cannot be located using the extensions saved with the report.

Data Type

Enumerated integer

Availability

Run time; design time

LastErrorCode

Description

Returns the error code for the last runtime error. It will be one of the following four values:

0 = No error;

1 = User canceled;

2 = Error in runtime parameters;

3 = Error in report.

Usage

[form.]*ControlName*.LastErrorCode

Example

```
'If error occurs, display error message
RRReport1.Action = 1
if RRReport1.LastErrorCode <> 0 then
     MsgBox  RRReport1.LastErrorString
end if
```

« If an error occurs, this code calls up a message box that displays the error string. »

Comments

LastErrorCode is only valid after setting the Action property to 1. If you set Action to 2, the report is run asynchronously, so LastErrorCode will not be set.

Data Type

Integer

Availability

Run time (read and write)

LastErrorPage

Description

Returns the page number of the last successfully printed page.

Usage

[form.]*ControlName*.LastErrorPage

Example

```
     'If error occurs, display error message
     RRReport1.Action = 1
     if RRReport1.LastErrorCode <> 0 then
        pagestr$ = "; last page printed was "
        + str(RRReport1.LastErrorPage)
        MsgBox  RRReport1.LastErrorString +
pagestr$
     end if
```

« If an error occurs, this code calls up a message box that displays the error string and the last page printed. »

Comments

LastErrorPage is only valid after setting the Action property to 1. If you set Action to 2, the report is run asynchronously, so LastErrorPage will not be set.

Data Type

Integer

Availability

Run time (read and write)

LastErrorString

Description

Returns the error string for the last runtime error.

Usage

[form.]*ControlName*.LastErrorString

Example

```
'If error occurs, display error message
RRReport1.Action = 1
if RRReport1.LastErrorCode < > 0 then
     MsgBox  RRReport1.LastErrorString
end if
```

« If an error occurs, this code calls up a message box that displays the error string. »

Comments

LastErrorString is only valid after setting the Action property to 1. If you set Action to 2, the report is run asynchronously, so LastErrorString will not be set.

Data Type

String

Availability

Run time (read and write)

LoadProperties

Description

LoadProperties is a method that can be used to update the custom control settings with the current report settings.

Usage

[form.]*ControlName*.LoadProperties( )

Example

```
RRReport1.LoadProperties()
```

«Updates the custom controls with the settings from the current report. »

Comments

This method is used to load all custom control properties with the values from the current report; use it to display or explicitly see all report properties.

Availability

Runtime

LowScope

Description

Specifies the low scope of the records to be used when the report is printed.

Usage

[form.]*ControlName*.LowScope[= Scope$]

Example

```
RRReport1.LowScope = "Jones"
```

« Specifies that the scope of records to be used should start at “Jones” when the report is printed. »

Comments

Set this property if you want to override the low scope value saved in the report. Be sure to set the Scope property to 2 to use the override. See **Scope** for a complete description of this feature.

Data Type

String

Availability

Design time; Run time

MasterIndex

Description

Specifies the master index to be used in the report.

Usage

[form.]*ControlName*.MasterIndex[= MasterIndex$, [IndexType$, IndexTag$]]

Example

```
RRReport1.MasterIndex = "c:\mis\sales.mdx, N,
CUSTID"
```

« Uses the file c:\mis\sales.mdx, a numeric index with a tag of CUSTID in the report. »

Comments

The MasterIndex property enables you to specify a master index for a report that was saved without one, override the master index saved with a report, or remove a saved master index from a report. To specify a master index or override the saved one, the specification can consist of any or all of three values in the following order:

```
<index>,<index type>,<tag>
```

In this specification, <index> represents the index file specification, which can be a complete path and file name, a directory, or a file name.

- If you specify both a directory and a file name, this directory is the only directory searched and this file name is the only file the Viewer searches for.
- If you specify a directory without a file name, the Viewer searches the specified directory for the master index name saved with the report.
- If you specify a file name without a directory, the Viewer searches for a file with the specified name in the directory of the master index saved with the report, then in the current master database directory, then in the default data directory specified via the DataDirectory property or in RRW.INI. If no default is specified, the Viewer searches for the file in the current directory.

In this specification, <index type> is the data type of the new index, represented as **N** for numeric, **C** for character, **D** for date, or **DT** for datetime. You can omit this value if you are overriding a saved master index and the replacement index is the same type (although it is good practice to include the data-type specifier). If the index named in the specification is a multiple-field index file (MDX, CDX, or WDX), the <tag> part of the specification represents an index tag. If you omit the <index> or <index type> part, use a comma as a place holder.

For example, the following specification replaces the master index saved with the report with an index named CUST96.MDX in the CUSTOMER directory on drive C. Since the replacement index is the same data type as the original index, the specification includes a comma in place of the <index type>. It also replaces the saved index tag with the CUSTID index tag:

```
C:\CUSTOMER\CUST96.MDX,,CUSTID
```

The following specification changes only the index tag of the master index saved with the report:

```
,,CUSTID
```

To remove the master index saved with the report without specifying a new master index, use the MasterIndex value **,R**.

If you omit this field or leave it blank, the Viewer uses the master index saved with the report, if any.

At design time, you can change this property in two ways:

- Double-click this property (or click the ellipsis button) to select an index.
- Simply enter the index, type, and tag into the settings box.

Data Type

String

Availability

Design time; Run time

MasterTable

Description

Specifies the name of a table that will override the master table saved with the report.

Usage

[form.]*ControlName*.MasterTable[= MasterTableName$]

Example

```
RRReport1.MasterTable = "c:\mis\sales.dbf"
```

« Uses the file c:\mis\sales.dbf as the master table for the report. »

Comments

This property is optional. The master table you specify should have the same columns as the master table originally used in the report.

If you leave this property blank, the Viewer uses the master table saved with the report.

At design time, you can change this property in two ways:

- Double-click this property to display the Database property page. Then click the ellipsis button next to Master Table to open the Select Master Table dialog (see Figure 4.4).

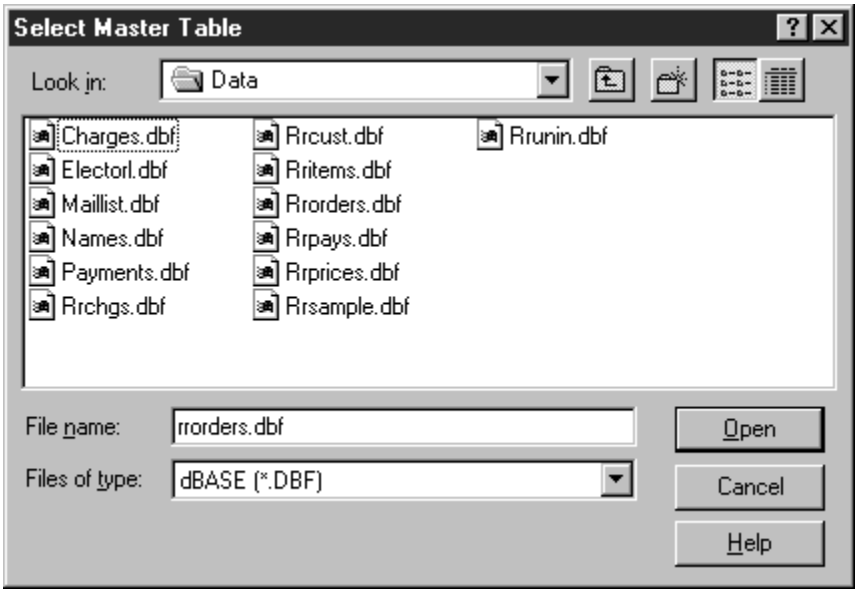

Figure 4.4 Select Master Table Dialog

- Simply enter the table name in the settings box.

Data Type

String

Availability

Design time; Run time

MemoFileName

Description

Specifies the name and optional directory location of the text memo file to be used in the report, which will override the text memo file saved with the report.

Usage

[form.]*ControlName*.MemoFileName[= MemoFileName$]

Example

```
RRReport1.MemoFileName = "c:\mis\q3notes.txt"
```

« Selects the memo file named “q3notes.txt” in the c:\mis directory. »

Comments

This property is optional.

- If both a directory and a file name are specified, this directory is the only directory searched and this file name is the only file the Viewer searches for.
- If a directory is specified without a file name, the Viewer searches the specified directory for the text memo file name saved with the report.
- If a file name is specified without a directory, the Viewer searches for a file with the specified name in the directory saved with the report, then in the default data directory as specified in the DataDirectory property or in RRW.INI.

If you leave this property blank, the Viewer uses the text memo file saved with the report, if any.

At design time, you can change this property in two ways:

- Double-click this property to see the Database property page. Then click the ellipsis button next to the Memo File box to display the Select Memo File dialog, which allows you to select a memo file and browse drives, directories, and files to which you have access.
- Simply enter the file name into the settings box.

Data Type

String

Availability

Design time; Run time

NoEscape

Description

Specifies whether a report can be canceled.

Usage

[form.]*ControlName*.NoEscape [= {True|False}]

Example

```
RRReport1.WindowNoEscape = True
```

« Specifies that a report cannot be canceled once it begins to be printed. »

Comments

This property is optional, and can be set to either True or False. True means the Cancel button in the status window is not active while reports are being output. False means the user may select Cancel during report output to pause or end the job. The default value is false.

Note that the Status window appears only when the DisplayStatus property is set to true.

If the user cancels synchronous execution of the report, the LastErrorCode property is set to C. If the report is run asynchronously, the RO_ECODE entry in the status file contains a C (see the section in Chapter 2 entitled **Understanding the Viewer Status File**.)

Data Type

Integer (Boolean)

Availability

Design time; Run time

Parameters

Description

Specifies user parameter values to be used when the report is printed.

Usage

[form.]*ControlName*.Parameters(ArrayIndex)
[= ParameterName$=ParameterValue$]

Enter a "name=value" string for each RIPARAM( ) function in your report for which you want to define a value. Use a separate line of code for each change.

The order of strings in the array does not matter, since each RIPARAM( ) function is identified by name.

Example

```
RRReport1.Parameters(0) = "Title=Cumulative
Earnings"
```

« Uses the value "Cumulative Earnings" wherever the function RIPARAM("Title") appears in the report. »

Comments

Use this property to define values for the RIPARAM( ) functions in your report. You can specify up to six (6) different parameters in the custom control (Parameters(0) – Parameters(5)). See the **Parameter Passing** section of Chapter 2 for information on using this feature.

Data Type

Array of strings

Availability

Run time

ParametersString

Description

Specifies user parameter values to be used when the report is printed.

Usage

[form.]*ControlName*.ParametersString[=ParameterName$=Parameter Value$]

Example

```
RRReport1.ParametersString = "Title=Cumulative
Earnings"
```

« Uses the value “Cumulative Earnings” wherever the function RIPARAM(“Title”) appears in the report. »

Comments

At design time, you can change this property array in two ways:

- Double-click this property to display the Parameters property page, which lists parameters and values in the report.
- Enter the parameter/value pairs separated by semicolons.

Data Type

Array of strings

Availability

Design time

Port

Description

Specifies the name of the printer port to which the report is to be printed.

Usage

[form.]*ControlName*.Port [= PortName$]

Example

```
RRReport1.Port = "LPT1:"
```

« Prints the report to the printer connected to “LPT1:". »

Comments

This property is optional. Enter a value such as “LPT1:” to override the printer port (and the printer associated with that port) saved with the report. Note that the colon is required.

You can also use the question mark (**?**) value or enter the word “Default” for this property. When the Port property contains a question mark, the user will see the Print dialog box shown in Figure 4.5. When

the Port property contains the word "Default," Viewer will use the default Windows printer port. (See the description of the **Printer** property.)

At design time, you can change this property by entering the port name into the settings box.

Data Type

String

Availability

Design time; Run time

Printer

Description

Specifies the name of the printer to which the report is to be printed.

Usage

[form.]*ControlName*.Printer [= PrinterName$]

Example

```
RRReport1.Printer = "HP LaserJet 4/4M"
```

« Prints the report to a printer called "HP LaserJet 4/4M." »

Comments

This property is optional. Enter a value to override the printer saved with the report. This property can have one of two values:

- The name of an available Windows printer. Available Windows printers are listed in the R&R Print dialog (accessed by selecting File ⇒ Print in interactive Report Writer). The value is case insensitive (that is, you can enter it in upper, lower, or mixed case).
- The question mark (**?**) value, to allow the user to select a printer at runtime. When the Printer property contains a question mark, the Print dialog will display, as shown in Figure 4.5.
- The word **Default** to force the Viewer to use the current default Windows printer. Use this setting only if you are sure that the default printer is compatible with the layout of your Viewer report(s).

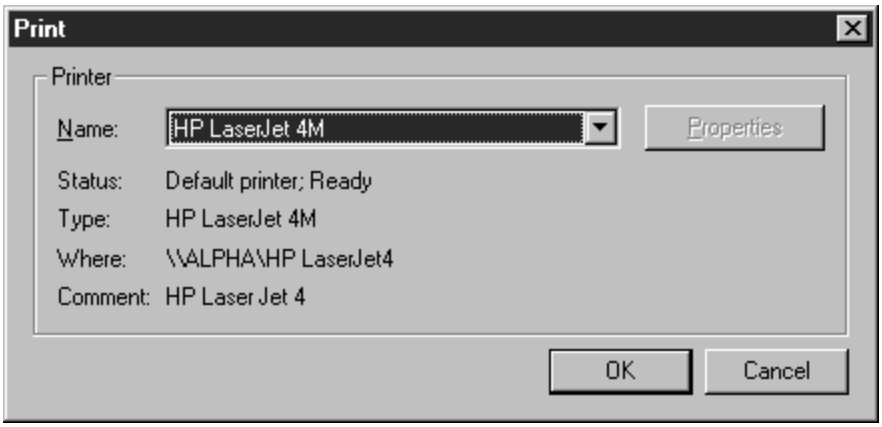

Figure 4.5 Print Dialog Box

Initially, the printer saved with the report is highlighted. The user can select another printer and port as necessary.

If this property is blank, the printer saved with the report will be used.

At design time, you can change this property in two ways:

- Double-click this property to display the Printer property page; then select the appropriate Printer Destination setting.
- Simply enter the printer name into the settings box.

Data Type

String

Availability

Design time; Run time

PrintFileName

Description

Specifies the name of the file to which the report is to be printed or exported.

Usage

[form.]*ControlName*.PrintFileName[= FileName$]

Example

```
RRReport1.PrintFileName =
"c:\output\q3sales.txt"
```

« Prints the report to a file named "q3sales.txt" in c:\output. »

Comments

Use this property if you have set the Destination property to 3, 4, 5, or 7 and you want to override the saved destination.

At design time, you can change this property in two ways:

- Double-click this property to display the PrintTo property page; then enter the appropriate file name in the File Name box.
- Simply enter the file name into the settings box.

Data Type

String

Availability

Design time; Run time

Query

Description

Specifies which, if any, query to use when the report is run.

Usage

[form.]*ControlName*.Query [= QueryOption%]

Example

```
RRReport1.Query = 2
```

« Ignores the query in the report and uses the query expression in the Filter property in place of it. »

Comments

The optional Query property allows you to control whether a query is applied to the report. Query can have one of four values:

0 – Saved. Run the report using the query saved with it, if any. The expression in the Filter property will be ignored and the report will be run exactly as it was saved.

1 – Entire. Run the entire report, ignoring any query saved in the report or contained in the Filter property.
2 – Override. Override the saved query with the expression in the Filter property. The report will be run with the records selected by the Filter property expression.
3 – Prompt user. Display a dialog box allowing the user to enter a query expression or edit the query saved with the report. If no query was saved with the report, the Insert Selection Rule dialog will display, as shown in Figure 4.6.

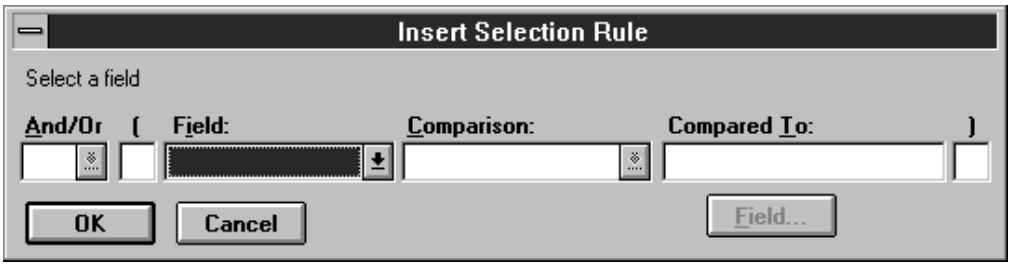

Figure 4.6 Insert Selection Rule Dialog Box

If a query was saved with the report, the Query dialog box will display, as shown in Figure 4.7.

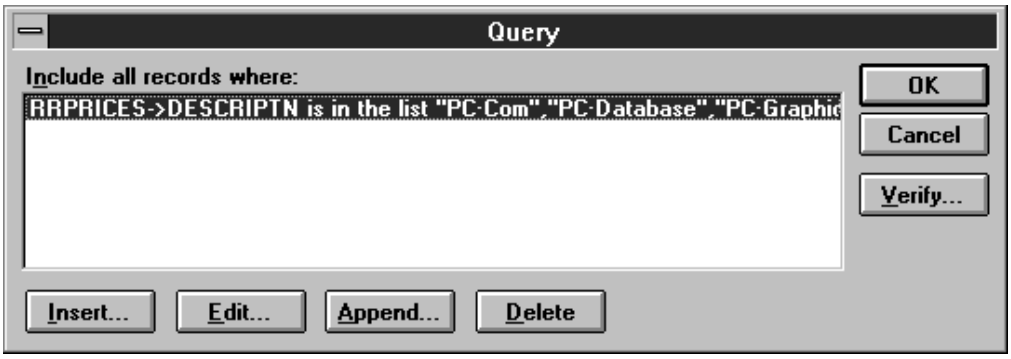

Figure 4.7 Query Dialog Box

When you set Query to 3, the value of the Filter property is always ignored.

Data Type

Integer (Enumerated)

Availability

Design time; Run time

RelatedTables

Description

Specifies related tables to override those saved with the report.

Usage

[form.]*ControlName*.RelatedTables(ArrayIndex)
[= Alias$=TableName$,<IndexName$>,<TagName$>]

Example

```
RRReport1.RelatedTables(0) =
"FIRST=c:\q2\first.dbf"
RRReport1.RelatedTables(1) =
"SECOND=c:\q2\second.dbf"
```

« Changes the first and second related tables in the report. »

Comments

These properties are optional. If you do not specify any related table overrides, the Viewer uses the tables saved with the report. It searches for these tables using the rules explained in Chapter 7.

When setting this property at run time, use a separate line of code for each change. Up to nine (9) different related tables may be specified (RelatedTables(0) – RelatedTables(8)).

Data Type

Array of strings

Availability

Run time

RelatedTablesString

Description

Specifies related tables to override those saved with the report.

Usage

[form.]*ControlName*.RelatedTablesString
[= Alias$=TableName$,<IndexName$>,<TagName$>]

Example

```
RRReport1.RelatedTablesString =
"FIRST=c:\q2\first.dbf;
SECOND=c:\q2\second.dbf"
```

« Changes the first and second related tables in the report. »

Comments

These properties are optional. If you do not specify any related table overrides, the Viewer uses the tables saved with the report. It searches for these tables using the rules explained in Chapter 7.

At design time, you can change this property array in two ways:

- Double-click this property to display the Relations property page. Then use the ellipsis buttons to select related tables and indexes. This is the preferred method, since it is easier and minimizes the possibility of syntax errors.
- Enter the related table entries separated by semicolons. If you want to override some related tables, but not all of them, you must use a semicolon as a place-holder. For example, to change the first and third related table, you would enter:

  “FIRST=c:\mis\first.dbf;;THIRD=c:\mis\third.dbf”.

Data Type

String

Availability

Design time

ReportDirectory

Description

Specifies a default directory where the Viewer may look for the report or library specified in ReportName or ReportLibrary.

Usage

[form.]*ControlName*.ReportDirectory[= DirectoryName$]

Example

```
RRReport1.ReportDirectory = "c:\mis\reports"
```

« Looks for the report in a directory called “c:\mis\reports.” »

Comments

If the report file name specified in the ReportName or ReportLibrary property does not contain full path information, then the Viewer will look for it in this directory. The default report directory you specify with this property will override any default report directory specified in the RRW.INI file.

Data Type

String

Availability

Design time; run time

ReportLibrary

Description

Specifies the library that contains the report to be printed.

Usage

[form.]*ControlName*.ReportLibrary[= LibraryFileName$]

Example

```
RRReport1.ReportLibrary =
"c:\rrw\rrsample\rrsample.rp5"
```

« Selects the report library named “rrsample.rp5” in the c:\rrw\rrsample directory. »

Comments

This property identifies the library that contains the report. The library name can include a path.

If you don’t include a path, the Viewer searches for the file in the default library directory specified in the ReportDirectory property. If no default is specified in ReportDirectory, the Viewer searches for the library in the default directory specified in the RRW.INI file. If

RRW.INI is not present and no default library directory is specified, the Viewer searches for the library in the current directory.

If you leave this property blank or if the library you specify cannot be found or read, the Viewer will return error status and, optionally, display an error message box (see **DisplayError**).

At design time, you can change this property in two ways:

- Double-click this property to display the General property page. Then click the ellipsis button next to Report Name to display the Open dialog, which allows you to select a report library file and browse drives, directories, and files.
- Simply enter the file name into the settings box.

Data Type

String

Availability

Design time; Run time

ReportName

Description

Specifies the name of the report to be printed.

Usage

[form.]*ControlName*.ReportName[= ReportName$]

Example

```
RRReport1.ReportName = "Order Invoice.RRW"
```

« Selects the report named “Order Invoice.” »

Comments

This property is required (unless ReportPick is set to 1 or 2). It contains the name under which the report was saved. If the report is being retrieved from a library, you must specify that library with the ReportLibrary property.

At design time, you can change this property in two ways:

- Double-click this property to display the General property page. Then click the ellipsis button next to Report Name to display

the Open Report dialog (see Figure 4.8), which contains a list of report files in the location specified in the ReportDirectory property.

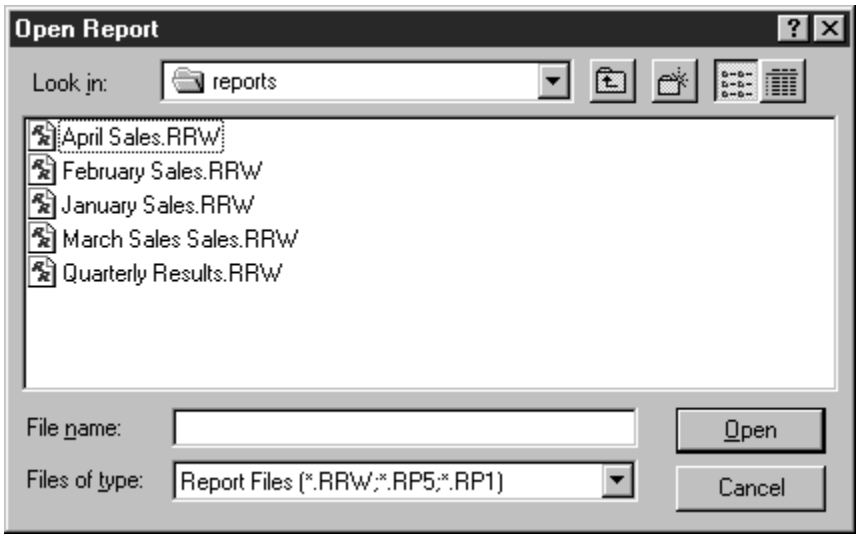

Figure 4.8 Open Report Dialog Box

- Simply enter the report name into the settings box.

Data Type

String

Availability

Design time; Run time

ReportPick

Description

Allows the user to pick one or more reports to be printed from a list of reports in the location specified by ReportDirectory.

Usage

[form.]*ControlName*.ReportPick [= PickOption%]

Example

```
RRReport1.ReportPick = 1
```

« Displays a list of reports and prints the highlighted report when the user selects OK. »

Comments

This property is optional, and can contain one of the following values:

0 – Pick none (use report in ReportName property);

1 – Pick one (allow user to select one report);

2 – Pick many (allow user to select several reports).

If you set this property to 1 or 2, you do not need to set the ReportName property; if you include both ReportPick and ReportName values, Viewer ignores the ReportName.

To have the Viewer prompt the user to select a succession of reports, set this property to 2 . When the value is 2, Viewer will prompt the user to select a report. After Viewer executes the selected report, the user will then be prompted to select another report. This prompt for report selection will repeat after each report until the user selects Cancel.

Set this property to 1 to prompt the user to select just one report. When the value is 1, Viewer will prompt the user to select a report (as with the 2 value), but will not prompt for an additional report selection after the report has been executed.

Data Type

Integer (Enumerated)

Availability

Design time; Run time

ResetControl

Description

Causes the control to reset all properties to their default states.

Usage

[form].RRReport1.ResetControl( )

Example

```
RRReport1.ResetControl( )
```

Comment

Use this method at runtime to clear all non-default values. It can be used to reset the control to a known state.

Data Type

Void

Availability

Design Time

ResetProperties

Description

Controls whether the custom control should reset its properties when a new report is specified.

Usage

[form].*ControlName*.ResetProperties[= {TRUE|FALSE}]

Example

```
RRReport1. ResetProperties = TRUE
```

Comment

Use this property to clear out values of a prior report. This setting causes *all* properties, except the properties visible from the General tab dialog, to be reset to their default states.

Data Type

Integer (Boolean)

Availability

Design time; Run time

RunReport

Description

RunReport is a method that can be used to trigger the print, display, or export of the report.

Usage

[form.]*ControlName*.RunReport(*action*)

Example

```
RRReport1.RunReport(1)
```

« Prints, displays, or exports the report, depending on the Destination property, and does not return until the report is completed. »

Comments

Use a value of 1 or 2 in for RunReport to print, display, or export the report in response to a user event. In most cases, it will be more convenient to set this property to 1.

If set to 1, the action is synchronous, which means that the next line of Visual Basic procedure code will not execute until the report is completed. The status of the report will be returned in the LastErrorCode, LastErrorString, and LastErrorPage properties.

If set to 2, the action is asynchronous, so that the report may still be running when the next line of Visual Basic procedure code is executed. When the report completes, its status is written into the status file.

Availability

Run time

Scope

Description

Specifies the scope of records to use when the report is printed.

Usage

[form.]*ControlName*.Scope [= ScopeOption%]

Example

```
RRReport1.Scope = 2
```

« Ignores the low and high scope saved in the report and uses the scope expressions in the LowScope and HighScope properties in place of them. »

Comments

This property is optional. Scope allows you to control the range of master table records that should be included in the report. You can specify a range of record numbers or index key values, ignore the scope saved with a report, or prompt the user to enter a range at run time. When you specify scope values, the Viewer reads only the records in the master table whose record number or index key is within the specified range. You can often speed up a report by using scope values with a master index. If you omit this field or leave it blank, the Viewer uses the saved scope values.

Scope can contain one of four values:

0 – Saved. Use the scope values saved with the report.
1 – Entire. Ignore any scope values.
2 – Override. Override the saved scope values with the values in the LowScope and HighScope properties.
3 – Prompt user. Allow the user to enter or change scope values at run time. When Scope is set to 3, the dialog box shown in Figure 4.9 displays. If WindowTitle is specified, the title bar will contain the WindowTitle. If WindowTitle is blank or missing, the title bar will contain the report name.

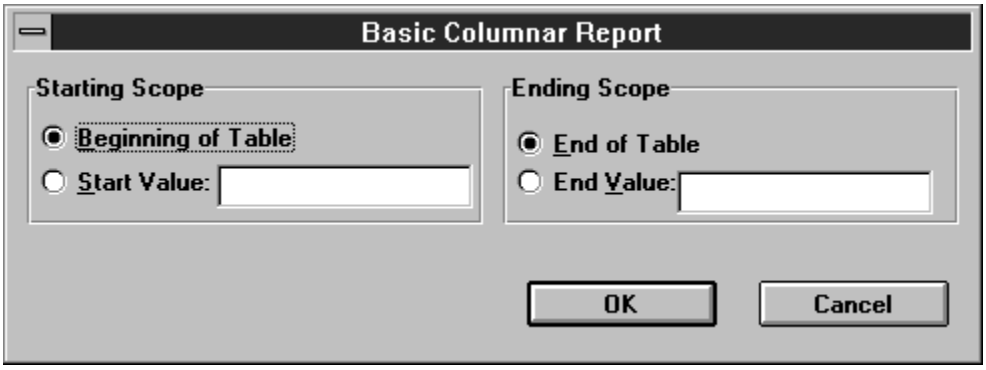

Figure 4.9 High and Low Scope Dialog Box

If Scope is set to 2, the LowScope property specifies the starting value of the scope and the HighScope property specifies the ending value of the scope. If Scope is set to 0, 1, or 3, the Viewer ignores LowScope and HighScope.

If no master index was saved with the report (or added using the MasterIndex property), the Viewer assumes the value is a record number. Otherwise, the Viewer assumes the value is a key value in the master index. In this case, the report begins reading the master table at the first record greater than or equal to the LowScope value and stops reading the master table after the last record found that is equal to or less than the HighScope value.

The range fully includes the end points. In other words, if you enter A as the low value and M as the high value, the Viewer reads the first record in which the value begins with A through the last record in which the value begins with M. For example, if you have a customer table indexed on last name and you want to print invoices for all customers whose name begins with a letter between A and M, enter 2 in Scope, the letter A in LowScope, and the letter M in HighScope.

All scope values must be character strings. Note that a date scope value must be in the format mm/dd/yy or mm/dd/yyyy.

Data Type

Integer (Enumerated)

Availability

Design time; Run time

SortFields

Description

Specifies the field(s) that are to be used to sort your data when the report is printed.

Usage

[form.]*ControlName*.SortFields(ArrayIndex)[= "+|-SortField$"]

Example

```
RRReport1.SortFields(0) = "+CUST.LNAME"
```

Comments

Sort fields can be database fields, calculated fields or total fields.

When setting this property at run time, use a separate line of code to specify each sort field. The first sort field you specify must be assigned array index 0, the second sort field must be assigned array index 1, etc.

The index values you assign must be continuous; no gaps are allowed (0,1,2 would be correct, but 0,1,3 would be wrong).

Data Type

Array of strings

Availability

Run time

SortFieldsString

Description

Specifies the field(s) that are to be used to sort your data when the report is printed.

Usage

[form.]*ControlName*.SortFieldsString[= "+|-SortField$"]

Example

```
RRReport1.SortFieldsString = "+CUST.LNAME"
```

Comments

Sort fields can be database fields, calculated fields or total fields.

At design time, you can change this property array in two ways:

- Double-click this property to display the Sort property page. Clicking on the down arrow next to each sort field will drop down a list of all fields used in the report from which you can select.
- Enter the sort field names separated by semicolons. If you want to override some sort fields, but not all of them, you must use a semicolon as a place-holder. For example, to change the first and third sort field, you would enter “Division;;Region”.

Data Type

String

Availability

Design time

StartPage

Description

Specifies the page of the report to start printing.

Usage

[form.]*ControlName*.StartPage[= Page%]

Example

```
RRReport1.StartPage = 10
```

« Specifies that the report should start printing at page 10. »

Comments

This property is optional. The StartPage and EndPage properties allow you to override the starting and ending page numbers saved with the report. The default value for these properties is blank.

To specify page numbers, include a StartPage value, an EndPage value, or both. If you specify both, EndPage must be equal to or greater than StartPage. For example, users can restart a canceled report where it was interrupted by specifying the starting page number as the StartPage value and 999999999 as the EndPage value. To reprint one or more consecutive pages of a report, specify the page numbers in the StartPage and EndPage properties. To print just one page, specify the same page number for both properties.

Data Type

Integer

Availability

Design time; Run time

StatusFileName

Description

Specifies the name and, optionally, the path for the Viewer status file.

Usage

[form.]*ControlName*.StatusFileName[=StatusFileName$]

Example

```
RRReport1.StatusFileName = "C:\TEMP\STATUS.TXT"
```

« Writes the status information into a file named STATUS.TXT in the TEMP directory on drive C. »

Data Type

String

Availability

Design time; Run time

SuppressTitle

Description

Specifies whether to suppress Title and Summary lines for a report that contains no records.

Usage

[form.]*ControlName*.SuppressTitle[={TRUE|FALSE}]

Example

```
RRReport1.SuppressTitle = FALSE
```

« Title and Summary lines will be printed even when no records are found. »

Comments

Set SuppressTitle to TRUE to suppress printing of Title and Summary lines if the report contains no records.

Data Type

Integer (Boolean)

Availability

Design time; Run time

TestPattern

Description

Specifies whether or not to print a test pattern showing the layout of the report on the page.

Usage

[form.]*ControlName*.TestPattern[= {True|False}]

Example

```
RRReport1.TestPattern = True
```

« Specifies that a test pattern of the report should be printed. »

Comments

This property is optional, and can be either True or False. True means to display a prompt before printing the report to allow the user the option of printing a test pattern. False means don't offer a choice to print a test pattern.

A test pattern is useful for aligning forms in the printer. The user can print the test pattern as many times as necessary and then print the report. If you enter **True**, the Viewer displays a box containing OK, Cancel, and Print buttons. The user can select OK and print as many test patterns as necessary to align the forms. Once the forms are aligned, the user can select Print to begin printing the actual report.

Data Type

Integer (Boolean)

Availability

Design time; Run time

UpdateControl

Description

Specifies whether the properties of the control should be updated with the properties of the report when a new report is selected.

Usage

[form.]*ControlName*.UpdateControl[= {True|False}]

Comments

Set this property to **True** if you want the current properties to reflect the current report.

Data Type

Integer

Availability

Design time; Run time

WindowBorderStyle

Description

Specifies the type of border for the preview window.

Usage

[form.]*ControlName*.WindowBorderStyle[= BorderStyle%]

Example

```
RRReport1.WindowBorderStyle = 2
```

« Sets a sizable border style for the preview window. »

Comments

Set this property to one of the following border styles if you are printing to a preview window (if Destination = 1).

1 – Fixed (a window of a fixed size with a standard border);
2 – Sizable (a window that can be resized by the user).

Note that for compatibility with earlier versions of R&R, this parameter accepts any of the following values:

0 (which formerly resulted in a borderless preview window) results in a fixed-size window with a standard border.
1 results in a fixed-size window with a standard border.
2 results in a variable-size window with a standard border.
3 (which formerly resulted in a fixed-size window with a double-line border) results in a fixed-size preview window with a standard border.

Data Type

Integer (Enumerated)

Availability

Design time; Run time

WindowControlBox

Description

Specifies whether the preview window is to have a control (system menu) box in the upper left corner when the report is printed to a preview window.

Usage

[form.]*ControlName*.WindowControlBox[= {True|False}]

Example

```
RRReport1.WindowControlBox = True
```

« Specifies that a control box (system menu) is to appear in the preview window. »

Comments

Set this property to True if you are printing to a preview window (if Destination = 1) and if you want the window to contain a control box.

Data Type

Integer (Boolean)

Availability

Design time; Run time

WindowHeight

Description

Sets the height of the window when the report is printed to a preview window.

Usage

[form.]*ControlName*.WindowHeight[= Height%]

Example

```
RRReport1.WindowHeight = 300
```

« Sets the height of the preview window to 300 pixels, or about 3 inches on most displays. »

Comments

The value for this property is expressed in pixels. Set this property if you are printing to a preview window (if Destination = 1).

Data Type

Integer

Availability

Design time; Run time

WindowLeft

Description

Sets the distance, in pixels, that the preview window is to appear from the left edge of the screen.

Usage

[form.]*ControlName*.WindowLeft[= Distance%]

Example

```
RRReport1.WindowLeft = 100
```

« Sets the left edge of the preview window 100 pixels from the left edge of the screen, about one inch on most displays. »

Comments

The value for this property is expressed in pixels. Set this property if you are printing to a preview window (if Destination = 1).

Data Type

Integer

Availability

Design time; Run time

WindowMaxButton

Description

Specifies whether the preview window is to have a maximize button when the report is printed to a preview window.

Usage

[form.]*ControlName*.WindowMaxButton[= {True|False}]

Example

```
RRReport1.WindowMaxButton = False
```

« Specifies that no Maximize button is to appear in the preview window. »

Comments

Set this property to True if you are printing to a preview window (if Destination = 1), and you want the window to contain a maximize button.

Data Type

Integer (Boolean)

Availability

Design time; Run time

WindowMinButton

Description

Specifies whether the preview window is to have a minimize button when the report is printed to a preview window.

Usage

[form.]*ControlName*.WindowMinButton[= {True|False}]

Example

```
RRReport1.WindowMinButton = True
```

« Specifies that a Minimize button is to appear in the preview window. »

Comments

Set this property to True if you are printing to a preview window (if Destination = 1). and you want the window to contain a minimize button.

Data Type

Integer (Boolean)

Availability

Design time; Run time

WindowTitle

Description

Specifies the title you want to appear in the preview window title bar when the report is printed to a preview window.

Usage

[form.]*ControlName*.WindowTitle[= Title$]

Example

```
RRReport1.WindowTitle = "Revenue Summary"
```

« Sets the title of the preview window (the string that appears on the title bar) to "Revenue Summary." »

Comments

This property is optional. Set this property if you are printing to a preview window (if Destination = 1), to specify a report title (for example, "Quarterly Profits") to be displayed in the following places:

- The Title Bar of the Preview window;
- The Print Status window (if DisplayStatus = True);
- The title bars of the dialog box that displays when the Printer, Port, Destination, or Scope value is a question mark.

If this property is blank, the report name will be used for the title.

When setting this property at run time, make sure that the title is enclosed in quotes.

Data Type

String

Availability

Design time; Run time

WindowTop

Description

Sets the distance, in pixels, that the preview window is to appear from the top edge of the screen.

Usage

[form.]*ControlName*.WindowTop[= Distance%]

Example

```
RRReport1.WindowTop = 100
```

« Sets the top edge of the preview window 100 pixels from the top of the screen, or about one inch on most displays. »

Comments

The value for this property is expressed in pixels. Set this property if you are printing to a preview window (if Destination = 1).

Data Type

Integer

Availability

Design time; Run time

WindowWidth

Description

Specifies the width of the preview window in pixels.

Usage

[form.]*ControlName*.WindowWidth[= Width%]

Example

```
RRReport1.WindowWidth = 500
```

« Specifies a preview window 500 pixels wide, or about five inches on most displays. »

Comments

The value for this property is expressed in pixels. Set this property if you are printing to a preview window (if Destination = 0).

Data Type

Integer

Availability

Design time; Run time

WriteAllow

Description

Controls whether database users can modify the tables and indexes in use by reports.

Usage

[form.]*ControlName*.WriteAllow[={True|False}]

Example

```
RRReport1.WriteAllow=False
```

« Prevent other applications from writing to the databases in use by the report. »

Data Type

Integer (Boolean)

Availability

Design time; Run time

XbaseEditor

Description

Specifies the type of memo editor that was used to create database memos for reports.

Usage

[form.]*ControlName*.XbaseEditor[={True|False}]

Example

```
RRReport1.XbaseEditor = False
```

« Memo fields used in the report have been created using a memo editor other than the standard Xbase memo editor. »

Data Type

Integer (Boolean)

Availability

Design time; Run time

Chapter 5
R&R Open Scripting

Introduction

R&R for Windows provides an open architecture for report generation. Called "R&R Open Scripting," this new architecture allows developers to create application-specific reporting front ends that use R&R's sophisticated reporting engine for actual report generation. R&R Open Scripting allows any developer who can generate text files to communicate effectively with R&R.

You can take advantage of the R&R Open Scripting interface to generate a report in either of two ways. The Report Wizards make use of this script mechanism to pass a user-defined report specification from the R&R Wizards to the main Report Designer executable. In order to provide a custom interface for report creation, developers can configure Report Designer to integrate their own Windows executable (EXE) in place of the R&R Wizard application.

In addition to the Report Wizard interface, a report can be generated by passing a script file pathname on the Report Designer command line. When Report Designer is invoked in this way, the script file is read and the report is created automatically. Passing a script file on the command line requires that a master table pathname be specified in the Report section of the script file. See the **Script File Format** section in this chapter for information about the contents of the script file.

Explanation of R&R Open Scripting is presented in the following sections:

- Custom Report Wizards
- Script File Format
- Script Command-Line Argument (/S)
- Report Wizard Input File

Custom Report Wizards

The R&R Open Scripting interface allows developers to integrate their own intelligent front ends to their applications. Through this

mechanism you can provide a custom user interface to gather the information needed to generate a report. As you will see, integrating your own custom user interface into R&R is very simple.

Open Scripting can also be used to generate reports with the Viewer. You can create a script file and have it executed by the Viewer to define and run a new report. To do so, you would supply the script file name to the Viewer in any of the following ways:

- as the RI_REPORT field value in a Viewer control file;
- as an argument to the chooseReport function of the Viewer DLL;
- as an element of the ReportName property of the custom control.

Configuring the Custom Application

To replace the R&R Wizards program with your own, simply add the following settings to the R&R initialization file (RRW.INI) located in your Windows directory:

```
[Special]
WizardEXE=C:\RR\MYWIZARD.EXE
```

If the **[Special]** section already exists, add the **WizardEXE** keyword to the existing section.

Note that if a full pathname is not specified for the executable file, Windows will search for it in the following order:

1. Current directory;
2. Windows directory;
3. Windows System directory;
4. The Report Designer program directory;
5. The DOS path;
6. Directories mapped on a network.

To restore the built-in Report Wizards, simply remove the **WizardEXE** setting from the initialization file.

Invoking the Custom Application

As is the case with the built-in Report Wizards, a custom application is invoked from either the Report Designer startup dialog or from the File New dialog. (Note that the Options ⇒ Preferences dialog provides

"File New" settings that allow these dialogs to be bypassed). After Report Wizards is selected and a database table is chosen, Report Designer immediately invokes the application specified in the initialization file. While the custom application is active, Report Designer is disabled; it is enabled when the custom application terminates.

Three arguments are passed to the custom application on the command line. These arguments are separated by semicolons (;). The first is the pathname to an "input file" containing information used by the Wizards. The second argument is the pathname to the script file in which the report information is to be written. The third argument is the complete pathname, including drive specifier, of the master database table selected within Report Designer. The master database table passed on the command line must be used by the custom application as the basis for report generation. For example, the following is a valid command line to the custom application:

```
C:\WIN\TEMP\WZI.TMP;C:\WIN\TEMP\WZO.TMP;C:\RR\EMP.DBF;
```

The script file is read when the custom application terminates. After validating the contents of the script file, Report Designer generates the report and performs any actions specified, such as previewing the report. Report Designer will indicate any errors detected by displaying an error message box identifying the invalid line in the file.

Script File Format

The R&R script file format is similar to that of a Microsoft Windows initialization (INI) file. Script files are made up of a series of sections that contain keyword definitions. However, these script files may contain multiple instances of a particular keyword in a given section and duplicate instances of a given section name, which is generally not the case with Windows initialization files.

The general rules that apply to the format of script files are as follows:

- Commas are used to separate data on a given line. Therefore field names may not contain commas; in addition, decimal values (field locations, margins, etc.) must always use a period as the decimal point and all dimensions are in inches.
- Blank lines or lines beginning with a semicolon (considered comment lines) are ignored.

- The maximum line length is 300 characters. Line continuation is not supported.
- All Boolean values are set by specifying either T or F.
- Missing parameters that are optional, such as margins or page size, will be set to default report values. Missing field length parameters will be set to the values stored in the database table.
- All field definitions specified will be inserted on the layout.
- All field location values are defined in hundredths of an inch. Values are specified as absolute (1.00) or relative (+1.00). Relative field locations are based on the end of any previously defined field on the line. Absolute column positions should be computed based on the Pitch specification in the [System] section of the input file. (Take into account the left margin setting when defining field locations — for example, if the left margin is 0.5 inches and field location value is 1.00, the field will be placed 1.50 inches from the left edge of the page.)
- Field alignment is specified by numeric values ranging from zero to five: 0=Left; 1=Center; 2=Right; 3=Wrap Left; 4=Wrap Right; 5=Wrap Fully Justified. Values 0 through 2 may be applied to any field. Values 3 through 5 may be applied only to character, memo, or logical fields.
- Field locations are based on field alignment. The location for a left-aligned field is the left edge of the field. The location for a right-aligned field is the right edge of the field. The location for a center-aligned field is the center of the field. The maximum field location is 25.00.
- Page margin values are defined in hundredths of an inch.
- Lines within a particular band must be specified in the order in which they are to appear in the report.
- Calculations and totals must be defined prior to reference by other fields. Calculated and total field names must be unique.
- Optional field style is specified as a combination of the letters BUIS, which can be combined to indicate Bold, Italic, Underscore, and Strikeout.
- Total field expressions are specified by separating the four required parameters by commas: Type, Reset, Accumulation, Running.
- Commas are used as place holders and may be omitted when defaulting trailing parameters on a line.

- Scripts passed to R&R via the **/S** command-line argument must supply a master table name in the REPORT section.

The following section provides a breakdown of the script file support in this version of R&R.

Script File Sections and Keywords

The report sections and keywords used in this version of Report Designer are indicated on the following pages. All sections and keywords are optional, with the exception of the **MasterTable=** keyword in the Report section, which is required when a script file is passed to Report Designer on the command line. This keyword is ignored if specified in a script file generated by a custom application that replaces the R&R Report Wizards.

REPORT SECTION
[Report]
MasterTable=pathname

If Report Designer is called with a script file as a command-line argument, the MasterTable argument will be used as the table within the specified database to be used to generate the report described in the remainder of the script file.

ACTION SECTION

R&R *Open Scripting currently supports two menu "actions." These optional commands allow for printing or previewing the report immediately after it is generated. Only one menu action should be specified for each script. If more than one action is specified, only the first one will be performed.*

[Actions]
Menu=FilePrint
Menu=FilePrintPreview

PAGE FORMAT SECTION

[PageFormat]
PageSize=(0: Letter; 1:Legal; 2: Executive; 3: A4)
TopMargin=(Inches)
BottomMargin=(Inches)
LeftMargin=(Inches)
RightMargin=(Inches)

Landscape=(T or F)
InterlineSpacing=(T or F)
PreviewZoom=(0: Minimum; 1: Mid-level; 2: Maximum)

RECORD FORMAT SECTION

[RecordFormat]
AveryLabel=(Label Name)
RecordsAcross=(1 to 99)
RecordWidth=(Inches)
RecordHeight=(Inches)
PrintColsAcross=(T or F)

CompressRecordGroup=(T or F)
SuppressRecordLines=(T or F)
BeginLineOnSemi=(T or F)
HeadFootSummary=(T or F)
BreakRecordArea=(T or F)

SORT SECTION

Sort levels must be declared in contiguous order (no gaps). Sort fields will be copied to group fields up until an existing group field definition (see below) is encountered. Setting "SortOrderN=T" indicates ascending order.

[Sort]
SortField1=FieldName
SortOrder1=(T or F)
    through:
SortField8=FieldName
SortOrder8=(T or F)

GROUP SECTION

Group fields must be declared in contiguous order (no gaps).

[Group]
GroupField1=FieldName
    through:
GroupField8=FieldName

BAND LINE SECTION

Band lines are created by placing band sections into the script file. One band line will be created for each band line section that is entered. Fields are positioned on each line by placing the field

definition keywords (described below) within the appropriate band line section.

[Title]
[PageHeader]
[GroupHeader1] through [GroupHeader8]
[Record]
[GroupFooter1] through [GroupFooter8]
[PageFooter]
[Summary]

FIELD DEFINITION KEYWORDS

The following field definition keywords must be placed within the appropriate band line section. Field definitions are indicated by a leading keyword, such as "CharField=", followed by a series of parameters separated by commas.

Here is an example of a database character field definition that will be placed on a Record line. Field name is CUSTNAME*; trim is* True; *location is* 1.00 inches; *alignment is* left; *style is* Underscored; *and length is* zero *(use the field length specified in the master table)*.

Example: **[Record]**
CharField=CUSTNAME, T, 1.00, 0, U, 0

DATABASE CHARACTER FIELD

CharField=Name (required),
Trim (T or F),
Location,
Alignment,
Style (BUIS),
Length

DATABASE NUMERIC FIELD

NumField=Name (required),
Trim (T or F),
Location,
Alignment,
Style (BUIS),
Integers,
Decimals,
Numeric picture:
0: Fixed; 1: Scientific; 2: Currency; 3: Comma; 4: General;
5: Percent

DATABASE DATE FIELD
DateField=Name (required),
Trim (T or F),
Location,
Alignment,
Style (BUIS),
Date picture:

0: dd-mmm-yy	12: dd/mm/yy
1: dd-mmm-yyyy	13: dd/mm/yyyy
2: dd-mmm	14: dd.mm.yy
3: mmm-yy	15: dd.mm.yyyy
4: mmm-yyyy	16: yy-mm-dd
5: mmmm d, yyyy	17: yyyy-mm-dd
6: d mmmm yyyy	18: mm/dd
7: mmmm yyyy	19: dd/mm
8: mmmm d	20: dd.mm
9: d mmmm	21: mm-dd
10: mm/dd/yy	22: Long Regional
11: mm/dd/yyyy	23: Short Regional

DATABASE DATE/TIME FIELD
DateTimeField=Name (required),
Trim (T or F),
Location,
Alignment,
Style (BUIS),
Date picture:

0: dd-mmm-yy	12: dd/mm/yy
1: dd-mmm-yyyy	13: dd/mm/yyyy

2: dd-mmm
3: mmm-yy
4: mmm-yyyy
5: mmmm d, yyyy
6: d mmmm yyyy
7: mmmm yyyy
8: mmmm d
9: d mmmm
10: mm/dd/yy
11: mm/dd/yyyy
14: dd.mm.yy
15: dd.mm.yyyy
16: yy-mm-dd
17: yyyy-mm-dd
18: mm/dd
19: dd/mm
20: dd.mm
21: mm-dd
22: Long Regional
23: Short Regional

Time picture:

0: h:mm
1: hh:mm
2: h:mm:ss
3: hh:mm:ss
4: h:mm am
5: hh:mm am
6: h:mm:ss am
7: hh:mm:ss am
8: Regional

DATABASE TIME FIELD

TimeField=Name (required),
Trim (T or F),
Location,
Alignment,
Style (BUIS),

Time picture:

0: h:mm
1: hh:mm

2: h:mm:ss

3: hh:mm:ss

4: h:mm am

5: hh:mm am

6: h:mm:ss am

7: hh:mm:ss am

8: Regional

TEXT FIELD

TextField=Text (required, placed within double quotes),
Trim (T or F),
Location,
Alignment,
Style (BUIS)

CALCULATION CHARACTER

CalcChar=Name (required),
Trim (T or F),
Location,
Alignment,
Style (BUIS),
Length (required),
Expression:
Calculated Field: Standard expression format (within double quotes), or
Total Field: Type, Field, Reset, Accumulation, Running

Type	***Reset***	***Accumulation***	***Running***
0: Count	G: Grand	A: Automatic	T (Running)
3: Minimum	P: Page	E: Every	F (Preprocessed)
4: Maximum	1-8: Group	P: Page	
		1-8: Group	

CALCULATION NUMERIC

CalcNum=Name (required),
Trim (T or F),
Location,
Alignment,
Style (BUIS),

Integers (required),
Decimals (required),
Numeric picture:
0: Fixed; 1: Scientific; 2: Currency; 3: Comma; 4: General;
5: Percent
Expression:
Calculated Field: Standard expression format
(within double quotes) or,
Total Field: Type, Field, Reset, Accumulation, Running

Type	*Reset*	*Accumulation*	*Running*
0: Count	G: Grand	A: Automatic	T (Running)
1: Sum	P: Page	E: Every	F (Preprocessed)
2: Average	1-8: Group	P: Page	
3: Minimum		1–8: Group	
4: Maximum			
5: Standard Deviation			
6: Variance			

CALCULATION DATE

CalcDate=Name (required),
Trim (T or F),
Location,
Alignment,
Style (BUIS),
Picture (see DATABASE DATE FIELD above)
Expression:
Calculated Field: Standard expression format
(within double quotes) or,

Total Field: Type, Field, Reset, Accumulation, Running

Type	*Reset*	*Accumulation*	*Running*
0: Count	G: Grand	A: Automatic	T (Running)
3: Min	P: Page	E: Every	F (Preprocessed)
4: Max	1-8: Group	P: Page	
		1–8: Group	

CALCULATION DATE/TIME

CalcDateTime=Name (required),

Trim (T or F),

Location,

Alignment,

Style (BUIS),

Date picture (see DATABASE DATE/TIME FIELD above)

Time picture (see DATABASE DATE/TIME FIELD above)

Expression:

Calculated Field: Standard expression format
(within double quotes) or,
Total Field: Type, Field, Reset, Accumulation, Running

Type	***Reset***	***Accumulation***	***Running***
0: Count	G: Grand	A: Automatic	T (Running)
3: Min	P: Page	E: Every	F (Preprocessed)
4: Max	1-8: Group	P: Page	
		1-8: Group	

CALCULATION TIME

CalcTime=Name (required),

Trim (T or F),

Location,

Alignment,

Style (BUIS),

Time picture (see DATABASE TIME FIELD above)

Expression:

Calculated Field: Standard expression format
(within double quotes)

or,
Total Field: Type, Field, Reset, Accumulation, Running

Type	*Reset*	*Accumulation*	*Running*
0: Count	G: Grand	A: Automatic	T (Running)
3: Min	P: Page	E: Every	F (Preprocessed)
4: Max	1-8: Group	P: Page	
		1-8: Group	

POINTSIZE

PointSize=Size in points

R&R Open Scripting uses the default point size set in Options ⇒ Default Settings for all fields. You can use the PointSize keyword to specify a new point size. Include the PointSize keyword in any Band Line Definition section to set a point size for all subsequent fields specified until another PointSize keyword is encountered.

In the following example, PointSize=16.0 changes the point size to 16 for the DEPARTMENT text field and to 14 for the FULLNAME field. PointSize=0 then returns the point size to the default for the SALARY field.

```
;BAND LINE DEFINITION SECTION
[PageHeader]
PointSize=16.0
TextField="DEPARTMENT", F,0.0, 0, U
PointSize=14.0
TextField="FULLNAME", F,2.5, 0, U
PointSize=0
TextField="SALARY", F,6.0, 2, U
```

Sample Script Output

A sample label script generated by the R&R Report Wizards is shown below. Although the script is relatively simple, it generates a fully formatted report containing report parameters such as sort and group information, various report bands, total fields and calculated fields. Note that the [Group] section is not required, since the group settings are automatically copied from the sort settings. Because the **FilePrintPreview** keyword has been specified as the menu action in the [Actions] section, the report will be previewed automatically after it has been generated.

```
; REPORT SECTION
[Report]
MasterTable=c:\data\employee.dbf

; ACTION SECTION
[Actions]
Menu=FilePrintPreview

; PAGE FORMAT SECTION
[PageFormat]
PageSize=0
TopMargin=.5
BottomMargin=.5
LeftMargin=.5
RightMargin=.5

; BAND LINE DEFINITION SECTION
[Title]
TextField="DBRSAMPL", F, 0.00, 0
TextField="*** R&R Grouped Columnar Report
***",F,3.75,1
CalcChar=wizDate,F,7.5,2,,8,"DTOC(date())"
[Title]
[GroupHeader1]
CharField=DEPARTMENT,F,0.0,0,BI
[GroupHeader1]
TextField="DEPARTMENT",F,0.0,0,BU
TextField="FULLNAME",F,1.7,0,BU
TextField="HIRE_DATE",F,4.5,1,BU
TextField="SALARY",F,6.0,2,BU
[Record]
CharField=DEPARTMENT,F,0.0
CharField=FULLNAME,F,1.7
DateField=HIRE_DATE,F,4.5,1
NumField=SALARY,F,6.0,2
[PageFooter]
[PageFooter]
TextField="Page ",F,3,1,B
CalcNum=wizPage,T,4.5,2,B,3,0,0,"PageNo()"
[GroupFooter1]
CalcNum=wizGpTot3,F,6.0,2,,,,0,2,SALARY,1,A,T
[Summary]
CalcNum=wizGrTot3,F,6.0,2,,,,0,2,SALARY,G,A,T
```

```
; SORT SECTION
[Sort]
SortField1=DEPARTMENT
SortField2=FULLNAME
```

Script Command-Line Argument (/S)

A script file may be passed on the Report Designer command line by appending the pathname of the script file to the **/S** switch:

/SC:\RR\SCRIPT.TXT

When Report Designer starts, the script file will be opened and validated. If no errors are encountered the report will be generated and any actions requested, such as previewing the report, will be performed.

Note: When a script file argument is passed to Report Designer, command-line arguments **/L** (library path), **/R** (report name), **/T** (table name), and **/I** (table name/instant report) are ignored.

Report Wizard Input File

Report Designer creates a temporary input file that is used by the R&R Wizards. This input file contains information that is useful to users of the R&R Open Scripting interface. The input file format is similar to that of a Microsoft Windows initialization (INI) file. These files are made up of a series of sections that contain keyword definitions. (Although the information passed in the input file is used by the R&R Report Wizards, this file may be ignored by any custom application.)

The sections and keywords placed in the input file are indicated in the following paragraphs. Some sections and keywords will always be present; others are optional and depend on the number of records in the table passed to the Wizards.

```
[System]
Product=0 (RRW)
Pitch=12 (Pitch of default font)
PageWidth=7.50 (Default width minus left/right
margins)
```

The TableDef section lists the supported database fields in the table passed on the command line, including the data type and field lengths. For numeric fields, the integer and decimals places are specified.

The data types are as follows:

0: Character	4: Memo
1: Numeric	5: Date/Time
2: Date	6: Time
3: Logical	

The following is a sample TableDef section for a master table containing six fields. Note that the VALUE field is a numeric field and includes integer and decimal places instead of a single field length.

```
[TableDef]
NAME=0,21
STREET=0,17
CITY=0,12
STATE=0,2
ZIP=0,5
VALUE=1,5,0
```

Up to twenty rows of result data from the table passed to the Report Wizards are included in the input file. Each of these records is placed in a separate section, labeled Row1 through Row20. Data in each field is truncated to 50 characters. Here are samples of two rows from the database table described above:

```
[Row1]
NAME=Ashley, Steve
STREET=100 Main St
CITY=Westboro
STATE=MA
ZIP=01581
VALUE=38500
[Row2]
NAME=Axelhouse, Jim
STREET=201 Oak Ave
CITY=Northboro
STATE=MA
ZIP=01532
VALUE=25500
```

Chapter 6
Interfacing to Application DLLs

Introduction

R&R includes a special function, CDLL( ), that allows you to call a function in a Windows Dynamic-Link Library (DLL) from a report. You might use CDLL( ) when you want to write a DLL-based function to perform an operation that R&R's UDFs don't support, such as a trigonometric function. CDLL( ) also provides access to functions that are used by other elements of your application, since DLLs are available to all parts of a Windows application.

Syntax

CDLL( ) takes three string arguments and returns a string value. The syntax is:

```
CDLL(string1,string2,string3)
```

where **string1** is the name of the DLL that contains the function, **string2** is the name of the function, and **string3** is an argument being passed to the DLL function. (Note that in Windows 95 and Windows NT, DLL function names are case sensitive.) You can use R&R functions to convert the argument from other data types and to return other data types. For example, the calculated field expression

```
CDLL("CONVERTS.DLL","MILES_KILO",STR(DISTANCE))
```

uses the STR function to convert the decimal value of DISTANCE into a character string and passes the string value to the MILES_KILO function in CONVERTS.DLL, which converts the distance in miles to kilometers.

CDLL( ) expects a boolean return value from the called DLL function: true to indicate the function executed successfully, or false to indicate an error. If the DLL returns a false value, CDLL( ) returns an error string. If the DLL function executes successfully, it should overwrite its input string with the output string to be returned by the CDLL( ) function. R&R passes the input and output strings using an 8000-byte buffer.

Example

This example uses CDLL( ) to call the functions RR_SIN, RR_COS, and RR_TAN from TRIGS.DLL. The functions are used to return the sine, cosine, and tangent values of the field DEGREES. Since CDLL( ) takes an input string and produces an output string, we first created three UDFs that take the value of DEGREES as a numeric and return its value as a numeric. These values are converted to character strings before being passed to TRIG.DLL. The three UDFs and their declarations and formulas are:

```
SIN(N_DEGREES) =
   VAL(CDLL("TRIG.DLL","RR_SIN",STR(DEGREES,6,0)))
COS(N_DEGREES) =
   VAL(CDLL("TRIG.DLL","RR_COS",STR(DEGREES,6,0)))
TAN(N_DEGREES) =
   VAL(CDLL("TRIG.DLL","RR_TAN",STR(DEGREES,6,0)))
```

In each UDF formula, the STR function converts the numeric value of DEGREES into a character string, as required for the third argument to CDLL( ). The second argument of STR specifies the character length of the string; the third argument specifies the number of decimal places. The VAL function converts the string result of CDLL( ) into a numeric representation, which is more useful for such functions.

Creating these UDFs allows you to supply the DEGREES argument as a numeric value and return it as a numeric value; the conversion to string representation and back is "hidden." To use these UDFs to access the DLL functions, you create calculated fields whose expressions supply the DEGREES as numeric arguments to SIN( ), COS( ), and TAN( ). Note the following:

- Although you can pass only a single argument to a DLL, that string can contain multiple arguments that can be parsed by the DLL function. The single string value returned by the function can also contain multiple values that can be parsed within a calculated field expression.
- If you use CDLL( ) in reports you plan to distribute for use with the Viewer, make sure that the referenced DLLs are available when the Viewer is executed.

Chapter 7
Distributing Reports

Introduction

This chapter provides information that is particularly useful to application developers who are creating reports for distribution to other users, whether the reports are designed for use interactively or with the Viewer. Information on required files for distributing reports is presented in the following sections:

- Distribution Files for the Viewer Executable
- Distribution Files for the Viewer DLL
- Distribution Files for the OCX
- Retrieving Report Files
- Consistency Checking

Distribution Files for the Viewer Executable

Figure 7.1 lists required and optional files to be distributed in order to enable users to run reports with the Viewer executable. You can specify locations for some of these files either with command switches or with the Viewer control fields.

Note that Report Designer saves the location of tables, indexes, image files, and text memo files with the report. The Viewer will automatically find these files if they are in the saved locations. If they are not in the saved locations, the Viewer will look for them in the default directories specified on the Viewer command line or in the RRW.INI file, if it is available. If the report files are not in either of these locations, use the parameters in the Viewer control file to specify file locations.

If you make RRW.INI available to users, it should be placed in the user's Windows directory. The UDF library file, RR.UDF, and the required DLL files (except as noted) should be in the same directory as the Viewer executable, RRWRUN.EXE.

Note also that in order to export reports to Excel Chart or PivotTable, your users must have Excel 5.0 (or later) installed on their systems.

File	Description	Location
RRWRUN.EXE	The Viewer executable program. Required.	Program directory
RRPD.DLL	The R&R printer DLL. Required.	Program directory
SNBD6W9S.DLL	Image import file. Required if reports contain images.	Windows System directory
Library File(s)	One or more files that contain the reports to be run. The RI_LIBRARY control parameter must specify the library file. Required only if the reports to be run have been saved to a library.	Any directory
Runtime Control Table (and command line needed to run Viewer with it) or Text Control File	The database table or text file that provides the Viewer with information about the report(s) to be run. Required unless your application creates one at runtime.	Any directory
Data Files	The tables, indexes, text memo files, and image files used by your reports. All data files used by your report(s) are required.	Any directory
RRTXTX.DLL, RRDBFX.DLL, RRWKSX.DLL, RRRTFX.DLL, RRHTML.DLL, RRVIEWX.DLL	Required only if your Viewer reports have Export destinations of Text, Xbase, Worksheet, RTF, HTML, or ActiveX Viewer control, respectively.	Program directory
RRCSVX.DLL	Required only if your Viewer reports have Export destinations of Text Data/Word Merge.	
RRCHART.EXE, RRXCHART.DLL	Required only if your Viewer reports have Export destinations of Excel Chart.	Program directory
RRAUTO.EXE, RRXTAB.DLL	Required only if your Viewer reports have Export destinations of Excel PivotTable.	Program directory

Figure 7.1 Viewer Distribution Files (continued on next page)

File	Description	Location
GSW32.EXE, GSWAG32.DLL, GSWDLL32.DLL	Requuired only if the reports have embedded charts.	Windows System directory
RRPRVIEW.CAB	Required only if your reports have Export destinations of ActiveX Viwer control.	
RR.UDF	User-defined function file. Required if your Viewer reports use any user-defined functions.	Program directory
RRW.INI	The R&R configuration file; it is optional and can be customized for different users. If RRW.INI is in the Windows directory, Viewer uses the defaults defined in that file. However, command-line switches take precedence; any setting you specify using a command-line switch will always override the corresponding RRW.INI setting.	Windows directory
RRW.SRT	A text file that controls case sensitivity and character collation. Optional.	Program directory

Figure 7.1 Viewer Distribution Files

Distribution Files for the Viewer DLL

In addition to the required files listed in Figure 7.1, you must include RRRPT32.DLL when distributing reports for Viewer access using the Viewer DLL. Your user(s) should place this DLL in the Windows System directory.

Note that since the Viewer DLL communicates directly with the Viewer executable, a control table or file is not required unless you need one for use with the **getRuntimeRecord** function.

If the Viewer executable (RRWRUN.EXE) will not be in the same directory as the application that calls the Viewer DLL, modify the ProgDir32 setting in the [Defaults] section of RRW.INI. This setting identifies the R&R program directory.

For example, if you plan to have your user(s) place RRWRUN.EXE in D:\RRPROG, you would include the following entry in the RRW.INI file that you distribute:

```
[Defaults]
ProgDir32=D:\RRPROG
```

As a result, the Viewer DLL will be able to find RRWRUN.EXE regardless of the calling application's directory location.

Distribution Files for the OCX

In addition to the required files listed in Figure 7.1, you must include the following files when distributing reports for Viewer access using the R&R OCX:

- Your program executable;
- COMDLG32.DLL;
- MSVCRT20.DLL;
- MFSCAN32.DLL;
- OC30.DLL;
- RRRPT32.DLL;
- RRW32.OCX;
- THREED32.OCX.

Your user(s) should place the OCX and DLL files in the Windows System directory so that these files can be shared by applications in any directory.

Specifying the Location of RRWRUN.EXE

If the Viewer executable (RRWRUN.EXE) will not be in the same directory as your Visual Basic executable, modify the ProgDir32 setting in the [Defaults] section of RRW.INI. This setting identifies the R&R program directory.

For example, if you plan to have your user(s) place RRWRUN.EXE in D:\RRPROG, you would include the following entry in the RRW.INI file that you distribute:

```
[Defaults]
ProgDir32=D:\RRPROG
```

As a result, the OCX will be able to find RRWRUN.EXE regardless of the calling application's directory location.

Distributing VB Applications

You should use the Setup Wizard and Setup Toolkit that comes with Visual Basic to create a setup program for distributing your VB application. Refer to the Visual Basic documentation for information about building a setup program for your application.

When presented with a list of files to be distributed with your application, make sure that the DLLs and OCXs identified in the **Distribution Files for the OCX** section are included on this list. Also ensure that the description of RRW32.OCX in the [Files] section of the Wizard-generated SETUP.LST file includes the $(DLLSelfRegister) key value; for example:

```
File1=1,,RRW32.OC_,RRW32.OCX,$(WinSysPath),
$(DLLSelfRegister),$(Shared),4/1/1996,120000,1.
0.0.0
```

See the description of the [Files] section of SETUP.LST in your Visual Basic documentation for more details.

If you are not going to be using the Setup Wizard and Setup Toolkit to distribute your application, make sure that RRW32.OCX and the files needed by your application are installed on the user's system and registered in the user's registry.

Retrieving Report Files

Before R&R will display or print a report, it must locate all the files used in the report, including tables, indexes, text memo files, or image files. If R&R cannot locate these files it will return an error message when you try to display or print the report. In the Viewer, this error will appear in your output status table. Viewer follows the rules described in the following sections to save and locate files used in a report. If you are developing reports that will not be retrieved from the same drive/directory in which they were saved, you need to know these rules.

In the following rules, "master" drive/directory is the drive/directory where the master table is currently located; "default" drive is the drive

where the default data directory is located; and "saved" drive/directory/file name is the drive, directory, and name of any file as it was when the report was last saved.

Master Table

If the master table and report library were in the same directory when the report was saved, R&R searches for the master table in 1) the drive/directory in which the report library is currently located, 2) the saved drive:\saved directory of the master table, and 3) the default drive:\default directory.

If the master table and report library were in different directories when the report was saved, R&R searches for the master table in 1) the saved drive:\saved directory of the master table and 2) the default drive:\default directory.

Related Tables and Indexes

When you save a report, R&R follows a set of rules to save the name of each related table and index file used in the report. In the following rules, assume that the master table is in C:\DIR1.

1. If the file's drive and directory are the same as the master drive and directory, R&R saves only the file name in the report definition. For example, if the full path and name of an index file is C:\DIR1\FILE1.NDX, R&R saves only FILE1.NDX.

   When the report is retrieved, R&R tries to locate required files by searching 1) master drive:\master directory\saved file name and 2) default drive:\default directory\saved file name. In each directory searched for an index file, the saved index file name extension will be tried first, then the default index extension specified in RRW.INI or on the R&R command line.

2. If the file's drive is the same as the master drive, but the directories differ, R&R saves both the file name and the directory. For example, if a report uses an index in C:\DIR2 and the index file name is FILE2.NDX, R&R saves \DIR2\FILE2.NDX.

When the report is retrieved, R&R tries to locate required files by searching 1) master drive:\saved directory\saved file name, 2) master drive:\master directory\saved file name, and 3) default drive:\default directory\saved file name. In each directory searched for an index file, the saved index file name extension will be tried first, then the default index file name extension.

3. If the file's drive differs from the master drive, R&R saves the entire path and file name. For example, if the report uses an index file whose name is D:\DIR3\FILE3.NDX, R&R saves D:\DIR3\FILE3.NDX.

   When the report is retrieved, R&R tries to locate required files by searching 1) saved drive:\saved directory\saved file name, 2) master drive:\master directory\saved file name, and 3) default drive:\default directory\saved file name. In each directory searched for an index file, the saved index file name extension will be tried first, then the default index file name extension.

Text Memo Files

When you save a report, R&R saves the complete path and name of any attached text memo file. When you retrieve a report that uses a text memo file, R&R first looks for the file in the drive/directory saved with the report, then in the master drive/master directory, then in the default data directory.

Image Files

When you save a report, R&R records whether the image was saved in the same directory as the report library. If it was, R&R will look for the image file in the directory that contains the current report library. If the image was not saved in the same directory as the report library, R&R will look in the directory that contained the image when the report was saved. If Report Designer does not find the image in the library directory or the saved directory, it will look in the default image directory. If the Viewer does not find the image file in the library or saved directory, it will search the default image directory specified in RRW.INI or identified with the **/I** command switch.

Consistency Checking

When a report is retrieved, R&R checks to see whether the saved report is consistent with the current database and index files. While you are responsible for keeping your indexes up to date, R&R will notify you of other discrepancies between the report and the files it uses. For example, R&R checks to see whether you have changed the field names in any of the tables since you last saved the report.

The following database changes affect reports saved in R&R:

- Deleting a field;
- Deleting a linking field;
- Changing the name or data type of a field;
- Changing the width of a field;
- Changing the name of a file.

The following sections explain how R&R responds to these changes. In many cases, R&R notifies you of the inconsistency between the report and the database. In Report Designer these messages will appear on the display. In the Viewer they will appear in your output status table and, if RI_DISPERR is set to true, on the display.

You can then edit the retrieved report to accommodate the changes made to your tables. (Note that you will not be notified of changes in field width.)

Deleted Field

If you delete from a table any field that is used by a report, R&R notifies you that the field is missing when it retrieves the report. It erases the field from the report, as well as erasing any totals based on it. If the deleted field is used in a calculated field expression, the calculated field will appear in the Field Menu flagged with a question mark in front of it. If any flagged fields are used in your report, you will have to edit the fields' expressions before printing the report.

If you delete a field used in a query, you will be prompted to edit the query when you try to print the report.

If you delete a sort or group field, R&R also deletes the sort or group fields below it in the Sort-Group Table. For example, if you delete COMPANY from your table, both COMPANY and PRODUCT will be deleted from the following list of four sort fields:

1 STATE
2 CITY
3 COMPANY
4 PRODUCT

Your Sort-Group Table will then contain only the following fields:

1 STATE
2 CITY

If you have deleted fields used to group your report, you may need to edit your total fields so that they reset at the appropriate level.

Deleted Linking Field

If you delete a database field that is used as a linking field, R&R creates a "dummy" linking field so it can retrieve the report. The dummy field has the same name as the missing linking field, with the prefix "?_" (as in ?_NAME). When you open the report, R&R displays a message notifying you that the field is no longer in the database, then displays the message "Relation must be edited." Use the Database ⇒ Relations dialog to correct or remove each relation whose description is flagged with a question mark (?). You cannot print or preview the report until you have corrected or deleted all incomplete relations.

Changed Field Name or Data Type

If you change the name or data type of a database field, R&R behaves as if the original field had been deleted and a new field added: the original field is removed from the composite record structure, along with any fields that total it, and the new field is added to the composite record structure. Use Insert ⇒ Field to insert the new field. Any calculated field that uses the changed field will appear in the Field Menu flagged with a question mark. If any flagged fields are used in your report, you must edit the fields' expressions before printing the report.

Changed Field Width

If you change the width of a field used in the report, R&R does not automatically adjust the width of the field on the report layout or in any calculated fields that use the field. You must use Format ⇒ Properties to make any necessary adjustment. If you use the field more than once in a report, you must adjust each occurrence of the field individually.

Changed File Name

If the name of a file has been changed, R&R searches for the file using the search rules previously described. If it can't find the file under its old name, R&R then displays the path and name of the file it can't find. Press Enter; then select or enter the path and file name of the renamed file.

Save the Revised Report

You should save a report after correcting for database or file location changes. If you don't, you will have to repeat the corrections the next time you retrieve the report. If you're not sure of the corrections, save the report under a different name.

Appendix A
Viewer Equivalencies

Introduction

R&R provides three ways of accessing the Viewer: the Viewer executable, the Viewer DLL, and the R&R Custom Control. Figure A.1 shows the equivalencies among the Custom Control properties, DLL routines, and Viewer executable control parameters, as well as the default value for each.

Table of Viewer Equivalencies

Custom Control Property	*DLL Equivalent*	*Viewer EXE Equivalent*	*Default Value*
(About)			
Action	execRuntime	(Not applicable)	(Not applicable)
CopiesToPrinter	setCopies	RI_COPIES	Saved number
DataDirectory	setDataDir	/D	Value in RRW.INI
Destination	setOutputDest	RI_PRINTER	Saved destination
DisplayError	setDisplayErrors	RI_DISPERR	False
DisplayStatus	setDisplayStatus	RI_STATUS	False
EndPage	setEndPage	RI_ENDPAGE	Saved ending page
ExportDestination	setExportDest	RI_EXPDST	Display
Filter	setFilter	RI_FILTER	(Not applicable)
GroupFieldsString	setGroupField	RI_GROUP1, ...	Saved group fields
HighScope	setHighScope	RI_HISCOPE	End of file
ImageDirectory	setImageDir	/I	Value in RRW.INI
IndexExtension	setIndexExtension	/X	Value in RRW.INI
LastErrorCode	returned from execRuntime	RO_ECODE	(Not applicable)
LastErrorPage	returned from execRuntime	RO_PAGES	(Not applicable)
LastErrorString	returned from execRuntime	RO_EMSG	(Not applicable)
LowScope	setLowScope	RI_LOSCOPE	Beginning of file
MasterIndex	setMasterIndexInfo	RI_MINDEX	Saved value
MasterTable	setMasterTableName	RI_MASTER	Saved value
MemoFileName	setMemoName	RI_MEMO	Saved value

Figure A.1 Viewer Equivalencies

Custom Control Property	*DLL Equivalent*	*Viewer EXE Equivalent*	*Default Value*
NoEscape	setPreventEscape	RI_NOESC	False
ParametersString	setUserParam	User-defined parameters	Blank
Port	setPrinterPort	RI_WPORT	Saved printer port
Printer	setPrinter	RI_WPTR	Saved printer driver
PrintFileName	setOutputFile	RI_OUTFILE	Saved value
Query	setFilterUsage	RI_QUERY	Use saved query
RelatedTablesString	setRelationInfo	RI_ALIAS1, ...	Saved related tables
ReportDirectory	setLibraryDir	/R	Value in RRW.INI
ReportLibrary	setLibrary	RI_LIBRARY	Required
ReportName	chooseReport	RI_REPORT	Required
ReportPick	setReportPick	RI_REPPICK	Blank
Scope	setScopeUsage	RI_SCOPE	Saved scope values
SortFieldsString	setSortField	RI_SORT1, ...	Saved sort fields
StartPage	setBeginPage	RI_BEGPAGE	Saved starting page
(Not applicable)	setStatusEveryPage	RI_CHKTIME	R
StatusFileName	setStatusFileName	/O	RRUNOUT.DBF or RRUNOUT.OUT
SuppressTitle	setSuppressTitle	/H	False
TestPattern	setTestPattern	RI_TEST	False
WindowBorderStyle	setWinBorderStyle	RI_WBORDER	Sizable
WindowControlBox	setWinControlBox	RI_WCTRL	True
WindowHeight	setWinHeight	RI_WHEIGHT	Maximized
WindowLeft	setWinLeft	RI_WLEFT	Maximized
WindowMaxButton	setWinMaxButton	RI_WMAX	True
WindowMinButton	setWinMinButton	RI_WMIN	True
WindowTitle	setWinTitle	RI_WTITLE	Report name
WindowTop	setWinTop	RI_WTOP	Maximized
WindowWidth	setWinWidth	RI_WWIDTH	Maximized
WriteAllow	setWriteAllow	/W	True
Xbase Editor	setXbaseEditor	/E	True

Figure A.1 Viewer Equivalencies (continued)

Index

A

About, 129
ACCUISR5.DLL, 198
Action, 129
Action routines, 43
AllowW, 10
Asynchronous printing, 118

C

C programs
  calling Viewer from, 36
Calculated fields
  changing width of, 206
  deleted fields in, 204
Case sensitivity
  in control table parameters, 13
CDLL() function, 195
Changing
  data type of field, 205
  field names, 205
  field width, 206
  file names, 206
choosePrinter, 50
chooseReport, 50
COMDLG32.DLL, 200
Command file, 5, 11
Command switches, 5
  /AL, 9
  /B, 9
  /D, 7
  /E, 9
  /H, 9
  /I, 7
  /O, 8
  /R, 6
  /T, 6
  /W, 8
  /X, 7
Consistency checking, 204
Control tables
  errors in, 34
  parameter widths, 13
  parameters for, 12
  Predefined parameters, 14
  question mark field values, 108
  question mark parameter values, 39
  record number, 4, 35
  required parameters, 13
  user-defined fields, 39
Control tables and files
  required parameters, 13
Copies
  specifying number to print, 16
CopiesToPrinter, 130
Object Linking and Embedding Control; ActiveX control, 117

D

Data files
  directory locations of, 7, 197, 202
Data type
  changing, 205
DataDir, 10
DataDirectory, 130
Default data directory
  specifying with /D switch, 7
Default image directory
  specifying with /I switch, 7
Default index file extension
  specifying with /X switch, 8
Default report directory
  specifying with /R switch, 6
Defining parameters, 39
Deleting
  database fields, 204
  group fields, 205
  linking fields, 205
  sort fields, 205
Destination, 131
DisplayError, 133
Displaying errors, 17

Displaying reports, 23, 95, 132
DisplayStatus, 134
DLLs
  calling functions from, 195
  required, 197

E

EndPage, 134
endReport, 52
Error messages, 33, 34, 35
  displaying on screen, 17
  in status table, 34
Error-Handling routines, 49
execRuntime, 52
ExportDestination, 135
Exporting reports to text files, 22, 23, 95

F

Field width
  changing, 206
  in control table, 13
Fields
  changing name, 205
  changing width of, 206
  deleting from database, 204
  deleting linking field, 205
File write access, 8
Files
  changing names of, 206
  directory locations, 202
  directory locations of, 201
  search rules for, 201
Filter, 136
Forms
  printing on, 29
FoxPro Windows
  calling Viewer from, 37
Functions
  calling from DLLs, 195

G

getBeginPage, 54
getCopies, 55
getDisplayErrors, 55
getDisplayStatus, 56
getEndPage, 56
getErrorInfo, 57
getExportDest, 59
getFilter, 59
getFilterUsage, 60
getFirstFieldName, 60
getFirstFilteredFieldName, 61
getFirstGroupField, 62
getFirstRelationInfo, 63
getFirstSortField, 64
getFirstUserParam, 65
getHighScope, 66
getLibrary, 66
getLowScope, 67
getMasterIndexInfo, 68
getMasterTableName, 68
getMemoName, 69
getNewReportHandle, 70
getNextFieldName, 70
getNextFilteredFieldName, 71
getNextGroupField, 71
getNextRelationInfo, 72
getNextSortField, 73
getNextUserParam, 74
getOutputDest, 74
getOutputFile, 75
Get-parameter routines, 44
getPreventEscape, 75
getPrinter, 76
getPrinterPort, 77
getReportPick, 77
getRuntimeRecord, 78
getScopeUsage, 79
getStatusEveryPage, 79
getTestPattern, 80
getWinTitle, 80
Group fields
  deleting, 205
GroupFields, 138
GroupFieldsString, 138
Grouping
  overriding with Viewer, 18

H

HighScope, 139

I

Image files
  directory location, 7
  search rules for, 203
ImageDirectory, 140
ImgDir, 10
ImgExt, 10
Index files
  default file extension, 8
  directory locations of, 7, 197, 202
  overriding with Viewer, 15
Index tags
  overriding with Viewer, 15
IndexExtension, 140
IndExt, 10
Interrupting reports, 21, 149

L

LastErrorCode, 118, 141
LastErrorPage, 118, 142
LastErrorString, 118, 143
LibDir, 10
Linking fields
  deleting, 205
LoadProperties, 144
LowScope, 144

M

Master index
  overriding with Viewer, 20, 145
Master table
  directory location of, 202
  overriding with Viewer, 19
  search rules for, 202
MasterIndex, 145
MasterTable, 147
MemExt, 10
Memo editor
  specifying with /E switch, 9
MemoEd, 10
MemoFileName, 148
MFSCAN32.DLL, 200
MSVCRT20.DLL, 200

N

NoEscape, 149

O

OC30.DLL, 200
OCX, 117

P

Page numbers
  ending, 135, 167
  last completed, 35
  starting, 135, 167
  starting and ending, 16
Parameter passing, 38, 39, 40, 41, 107, 108
  defining parameters, 39
  incorporating values in reports, 40
  question mark parameter value, 39, 108
  using control table, 38
  using parameter table, 41
  using RIPARAM, 40
Parameter tables, 41
Parameters, 149
ParametersString, 150
PointSize, 191
Port, 151
PowerBuilder
  calling Viewer from, 38
Previewing reports, 23, 95, 132
PrevWinClr, 10
Print dialog, 152, 153
Printer, 152
  overriding, 98, 152
  overriding with Viewer, 29
  selecting, 98, 152
Printer port
  overriding, 99, 151
  overriding with Viewer, 29
  selecting, 99, 151
PrintFileName, 153

Printing
number of copies, 16
on forms, 29
selected pages, 16, 135, 167
test patterns, 29
to file, 22, 97
Prompting users for input, 39, 108

Q

Queries, 17, 136
and Viewer, 154
deleted fields in, 204
overriding, 155
overriding with Viewer, 17, 24
Query, 154
Question mark parameter value, 13, 22, 23, 24, 26, 27, 29, 30, 39, 108

R

Related tables
overriding with Viewer, 15
RelatedTables, 156
RelatedTablesString, 156
Report libraries
directory locations of, 6, 197
distributing, 198
specifying for Viewer, 19
Report title, 174
Report titles, 30
ReportDirectory, 157
ReportLibrary, 158
ReportName, 159
ReportPick, 160
Reports
changes to, 204
specifying for Viewer, 25, 159
ResetControl, 161
resetErrorInfo, 81
ResetProperties, 162
RI_ALIAS#, 15
RI_BEGPAGE, 16
RI_CHKTIME, 16
RI_COPIES, 16
RI_DISPERR, 17, 33
RI_ENDPAGE, 16
RI_EXPDST, 17
RI_FILTER, 17
RI_GROUP, 18
RI_HISCOPE, 26
RI_LIBRARY, 19, 198
RI_LOSCOPE, 26
RI_MASTER, 19
RI_MEMO, 20
RI_MINDEX, 20
RI_NOESC, 21
RI_OUTFILE, 22
RI_PRINTER, 22
RI_QUERY, 24
RI_REPORT, 25
RI_REPPICK, 26
RI_SCOPE, 26
RI_SORT, 28
RI_STATUS, 28
RI_TEST, 29
RI_WBORDER, 31
RI_WCTRL, 32
RI_WHEIGHT, 32
RI_WLEFT, 32
RI_WMAX, 32
RI_WMIN, 32
RI_WPORT, 29
RI_WPTR, 29
RI_WTITLE, 23, 30, 40
RI_WTOP, 32
RI_WWIDTH, 33
RIPARAM, 39, 40
RO_ECODE, 21, 34
RO_EMSG, 34
RO_PAGES, 16, 35
RO_REPORTS, 35
RO_RIRECNO, 35
RR.UDF, 199
RRAUTO.EXE, 198
RRCHART.EXE, 198
RRCSVX.DLL, 198
RRDBFX.DLL, 198
RRPD.DLL, 198
RRRPT32.DLL, 200
RRRTFX.DLL, 198

RRTXTX.DLL, 198
RRUNOUT.DBF, 33
  directory location of, 8, 105
  fields, 34
  specifying alternate name for, 8, 105
  update frequency, 16
RRW.INI, 10, 119, 197, 199
RRW.SRT, 13, 199
RRW32.OCX, 200
RRWKSX.DLL, 198
RRWRUN.EXE, 2, 198
RRWRUN.OUT, 118
RRXCHART.DLL, 198
RRXTAB.DLL, 198
RTF export, 132
RunReport, 162
Runtime
  and queries, 154
  calling from C programs, 36
  calling from FoxPro, 37
  calling from Visual Basic, 37
  command switches, 5, 6
  control parameters, 12
  control tables, 13, 15, 16, 17, 19, 20, 21, 24, 25, 26, 27, 30, 103, 146, 163, 164
  displaying status window, 28
  interrupting reports, 21
  overriding indexes and tags, 15
  overriding master index, 20, 145
  printing test pattern, 29
  required DLLs, 197
  required files, 198, 199
  required parameters, 13
  specifying a range of records, 26, 163
  specifying master table, 19
  specifying number of copies, 16
  specifying queries, 24
  specifying report library, 19
  specifying report title, 30
  specifying reports, 25
  specifying starting and ending page numbers, 16
  specifying text memo files, 20
  status table, 33, 34, 35
  system requirements, 2
  using on a network, 9
  using RRW.INI for defaults, 10
Runtime distribution files, 198, 199
Runtime overrides
  Grouping, 18
  indexes and tags, 15
  master index, 20, 145
  master table, 19
  output destination, 22
  output file, 22
  printer, 29, 98, 152
  printer port, 29, 99, 151
  queries, 24, 136
  related tables, 15
  report output file, 97
  saved filter, 155
  scopes, 26, 163
  Sorting, 28
  text memo files, 20, 148
Runtime status window
  cancel button, 149
  displaying, 134

S

Saving reports as files, 22, 97
Scope, 163
Scopes, 26, 163
Search rules
  for image files, 203
  for master table, 202
  for related tables and indexes, 202
  for text memo files, 203
Select Master Table dialog, 147
Selecting
  ending page number, 135, 167
  starting page number, 135, 167
setBeginPage, 81
setCopies, 82
setDataDir, 82
setDisplayErrors, 83
setDisplayStatus, 84

setEndPage, 84
setExportDest, 85
setFilter, 85
setFilterUsage, 86
setGroupField, 88
setHighScope, 88
setImageDir, 89
setIndexExtension, 89
setLibrary, 90
setLibraryDir, 91
setLowScope, 92
setMasterIndexInfo, 92
setMasterTableName, 93
setMemoName, 94
setOutputDest, 95
setOutputFile, 96
Set-Parameter routines, 46
setPreventEscape, 97
setPrinter, 98
setPrinterPort, 99
setRelationInfo, 100
setReportPick, 101
setScopeUsage, 102
setSortField, 103
setStatusEveryPage, 104
setStatusFileName, 105
setSuppressTitle, 105
setTestPattern, 106
setUserParam, 106
setWinBorderStyle, 109
setWinControlBox, 109
setWinHeight, 110
setWinLeft, 111
setWinMaxButton, 111
setWinMinButton, 112
setWinTitle, 96, 108, 112
setWinTop, 113
setWinWidth, 114
setWriteAllow, 114
setXbaseEditor, 115
Shortcut Maker utility, 4
ShowSplash, 10
Sort fields
  deleting, 205
SortFields, 165
SortFieldsString, 165
Sorting
  overriding with Viewer, 28
Specifying Viewer reports, 159
StartPage, 166
Status table, 33, 34, 35
  control table record number, 35
  directory location of, 8, 33, 105
  error codes, 34
  error messages, 34
  fields, 34
  naming, 8, 105
  number of pages completed, 35
  number of reports, 35
  specifying with /O switch, 8
  update frequency of, 16
Status window
  displaying, 28
StatusFileName, 167
SuppressTitle, 168
Synchronous printing, 118

T

Test patterns
  printing, 29
TestPattern, 168
Text control files
  parameters, 12
  required parameters, 13
  structure of, 11
Text files
  exporting reports to, 22, 23, 95
Text memo files, 20, 148, 198
  directory locations of, 7, 197
  overriding, 148
  overriding with Viewer, 20
  search rules for, 203
THREED32.OCX, 200

U

UpdateControl, 169
User-Interface routines, 48
Using RRW.INI for defaults, 119

V

Viewer
command syntax for, 4
Visual Basic
calling Viewer from, 37

W

WindowBorderStyle, 169
WindowControlBox, 170
WindowHeight, 171
WindowLeft, 172
WindowMaxButton, 172
WindowMinButton, 173
WindowTitle, 132, 174
WindowTop, 174
WindowWidth, 175
WinExec, 37
WriteAllow, 176
writeRuntimeRecord, 115

X

XbaseEditor, 176