
ARPEGGIO
Information Publisher
Developing
Applications
CONCENTRIC™
DATA SYSTEMS

Version 2

Concentric Data Systems, Inc.
110 Turnpike Road
Westborough, MA 01581

Printed in the United States of America

Trademarks and Acknowledgments

Documentation Part Number: ARP2MA(2000-01)

Table of Contents

Chapter 1 Overview 1

Introduction 1
Organization of the Manual 1
Viewer Requirements 2

Chapter 2 Using the Report Viewer 3

Introduction 3
Executing the Viewer 3
Providing Viewer Input 4
The Viewer Command Line 4
Command Switches 6
Using RSW.INI for Default Information 11
Using Control Tables and Files 12
Creating a Control Table 12
Creating a Text Control File 12
Specifying Control Parameters 13
Parameters for Modifying Report Characteristics 14
Parameters to Control Viewer Preview Display 31
Understanding the Viewer Status File 33
Status File Entries 34
Application Calls to the Report Viewer 36
Calling the Viewer from C 36
Calling the Viewer from Visual Basic 37
Calling the Viewer from PowerBuilder 38

Chapter 3 Parameter Passing 39

Introduction 39
Passing Control Parameter Values 39
Defining Parameters 40
Prompting for User Input 40
Using a Parameter Table 42

Chapter 4 Accessing the Viewer DLL 45

Introduction 45
Action Routines 46
Get-Parameter Routines 46
Set-Parameter Routines 48
User-Interface Routines 50

Error-Handling Routines....51
Functions Provided by the Viewer DLL....51
chooseDataSource....51
choosePrinter....52
chooseReport....53
chooseTable....55
endReport....56
execRuntime....57
getBeginPage....58
getCopies....59
getDataSource....60
getDisplayErrors....60
getDisplayStatus....61
getEndPage....61
getErrorInfo....62
getExportDest....64
getFilter....64
getFilterUsage....65
getFirstFieldName....65
getFirstFilteredFieldName....66
getFirstGroupField....67
getFirstJoinInfo....68
getFirstReplace....69
getFirstSortField....70
getFirstUserParam....70
getLibrary....71
getMasterTableName....72
getMemoName....72
getNewReportHandle....73
getNextFieldName....73
getNextFilteredFieldName....74
getNextGroupField....75
getNextJoinInfo....75
getNextReplace....76
getNextSortField....77
getNextUserParam....78
getOutputDest....78
getOutputFile....79
getPreventEscape....79
getPrinter....80
getPrinterPort....81
getReportPick....81

getRuntimeRecord 82
getStatusEveryPage 83
getTestPattern 84
getWinTitle 84
resetErrorInfo 85
setBeginPage 85
setCopies 86
setDatabase 86
setDataDir 87
setDataSource 87
setDisplayErrors 88
setDisplayStatus 88
setEndPage 89
setExportDest 90
setFilter 90
setFilterUsage 91
setGroupField 92
setImageDir 93
setJoinInfo 94
setLibrary 94
setLibraryDir 95
setMasterTableName 96
setMemoName 96
setOutputDest 97
setOutputFile 99
setPassword 100
setPreventEscape 100
setPrinter 101
setPrinterPort 102
setReplace 103
setReportPick 104
setSortField 105
setStatusEveryPage 106
setStatusFileName 106
setSuppressTitle 107
setTestPattern 107
setUserName 108
setUserParam 109
setWhere 111
setWinBorderStyle 112
setWinControlBox 113
setWinHeight 113

setWinLeft 114
setWinMaxButton 114
setWinMinButton 115
setWinTitle 115
setWinTop 116
setWinWidth 117
writeRuntimeRecord 117

Chapter 5 Using the Custom Control (OCX) 119

Introduction 119
Installation 119
Determining Report Status 120
Using RSW.INI for Default Information 121
Using the OCX 121
Changing Values Using the Properties List 122
Changing Values Using the Control Properties Dialog 123
Custom Control Properties 129
(About) 129
Action 129
CopiesToPrinter 130
Database 131
DataDirectory 131
DataSource 132
Destination 133
DisplayError 135
DisplayStatus 136
EndPage 136
ExportDestination 137
Filter 138
GroupFields 140
GroupFieldsString 141
ImageDirectory 142
Include 142
LastErrorCode 144
LastErrorPage 145
LastErrorString 145
LoadProperties 146
MasterTable 147
MemoFileName 148
NoEscape 149
Parameters 150

ParametersString....150
Password....151
Port....152
Printer....153
PrintFileName....154
RelatedTables....155
RelatedTablesString....155
Replace....156
ReportDirectory....159
ReportLibrary....160
ReportName....161
ReportPick....162
ResetControl....163
ResetProperties....164
RunReport....164
SortFields....165
SortFieldsString....166
StartPage....167
StatusFileName....167
SuppressTitle....168
TestPattern....169
UpdateControl....169
UserName....170
Where....170
WindowBorderStyle....172
WindowControlBox....173
WindowHeight....173
WindowLeft....174
WindowMaxButton....174
WindowMinButton....175
WindowTitle....176
WindowTop....176
WindowWidth....177

Chapter 6 ARPEGGIO ReportScript....179

Introduction....179
Custom Report Wizards....180
Configuring the Custom Application....180
Invoking the Custom Application....181
Script File Format....181
Script File Sections and Keywords....183
Sample Script Output....191

Script Command-Line Argument (/S).....192
Report Wizard Input File.....192

Chapter 7 Interfacing to Application DLLs.....195

Introduction 195
Syntax.....195
Example 195

Appendix A Viewer Equivalencies 207

Introduction 207
Table of Equivalencies 207

Index.....209

Chapter 1
Overview

Introduction

This manual explains how to incorporate reports into your Windows applications, whether you are using the Report Viewer or distributing reports for use with 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 Report Designer.

Organization of the Manual

Chapters 2, 4, and 5 of this manual explain three methods for accessing the Report Viewer:

- You can directly access the Viewer executable (RSWRUN.EXE) using a control table or file. This method is explained in Chapter 2, "Using the Report Viewer."
- The Viewer DLL provides an Application Programming Interface (API) that is suitable for use by any high-level programming language. See Chapter 4, "Accessing the Viewer DLL," for details.
- The Viewer OCX (custom control) simplifies Report Viewer access for applications. Chapter 5, "Using the Custom Control," explains this method.

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

- Chapter 6, "ARPEGGIO ReportScript," explains how developers can pass a user-specified report specification to Report Designer by means of a script file.
- Chapter 7, "Interfacing to Application DLLs," explains use of ARPEGGIO's CDLL( )function to call a Windows Dynamic-Link Library (DLL) function from a report.
- Chapter 8, "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.

Viewer Requirements

To run the Viewer, you need the following:

- The Viewer program (RSWRUN.EXE), which is installed in the ARPEGGIO program directory if you choose to install the optional Viewer files during Setup.
- All Viewer distribution files required for your particular application. See Chapter 8, "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 Report Viewer

Introduction

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

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

Executing the Viewer

To use the Report Viewer to run an ARPEGGIO report, follow these general steps:

1. In ARPEGGIO Report Designer, 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, as well as any parameters you want to change at report execution. 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 Report Shortcut Maker utility to create program icons 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, you use either a database table (referred to as a *control table*) or a text file (referred to as a *control file*) to specify each report you want to run and any parameters you want to modify at report execution. For example, you can use the control table or file to override the SELECT statement for the report, the filter saved with the report, or the report's output destination.

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

The Viewer Command Line

To execute the Viewer, use the command **RSWRUN** along with the **/TS** or **/TT** switch, which identifies your control table or file. For example, to run the reports specified in the text control file REPORTS.TXT, you would use this command:

```
RSWRUN /TTreports.txt
```

The Viewer command line can also include other command switches listed in Figure 2.1 and described in the **Command Switches** section of this chapter. See the description of each command switch for details.

Command Line Using a Control Table

If you are using a database table to provide Viewer control parameters, you can generate all the reports specified in the control table by issuing the Viewer command with the **/TS** switch, which identifies the control table, and the **/CS** switch, which identifies the control table data source. For systems that require a user name and password for access to tables, as necessary you also need to use the **/CU**, **/CP**, **/U,** and **/P** switches, which provide passwords for the control table or report

tables. The Viewer command can also include one or more of the optional command switches listed in Figure 2.1. The syntax is:

```
RSWRUN /TS<table name> /CS<data source name> [switches]
```

Since the **/CS** switch identifies the control table data source, you can create a control table or file using any data source, regardless of which data source your Viewer reports will be accessing. For example, you can use a control table created in dBASE to run reports that use data from an Oracle database.

If you do not want to run all the reports specified in the control table, you can generate specific reports using one or more Row ID numbers with the Viewer command. The syntax is:

```
RSWRUN /TS<table name> [Row ID]...[switches]
```

Replace **<Row ID>** with the number assigned to the report you want to run. This Row ID number must match a value in the RI_ID column in your control table. (See the description of the RI_ID parameter.) You can specify multiple Row ID numbers on the command line or assign the same RI_ID value to multiple reports in the control table. To generate all the reports specified in the control table, omit the Row ID argument.

For example, to run all the reports specified in the SQL Server REPORTS table in the PAYROLL database on the ACCTS server whose RI_ID value is 3, issue this Viewer command:

```
RSWRUN /TSreports 3 /CSaccts /CDpayroll /CUjoe /CPsystem
```

To run all the reports specified in the REPORTS table in the PAYROLL database on the ACCTS server whose RI_ID value is 1 or 3, issue this Viewer command:

```
RSWRUN /TSreports 1 3 /CSaccts /CDpayroll /CUjoe /CPsystem
```

Command Line Using a Text Control File

If you are using a text file to provide Viewer control parameters, you generate a single report by issuing the Viewer command with the **/TT** switch followed by the file name and any optional command switches:

```
RSWRUN /TT<file name>  [switches]
```

The **/TT** switch is required with the file name, but all other switches are optional. You can include a path with the file name.

For example, the following command will run the report specified by the RUNIN.TXT file in the \DATA directory on the C: drive:

```
RSWRUN /TTc:\data\runin.txt
```

To run multiple reports with a single Viewer command, 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, precede the command file name with the @ symbol on the Viewer command line.

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

```
RSWRUN @REPORTS.CMD
```

As a result, the Viewer will run the reports specified in the control files listed in REPORTS.CMD.

The command file name can be followed by any command switches you want to specify. For example, to specify the user name “john” and password “accounts,” you would use this Viewer command:

```
RSWRUN @REPORTS.CMD /Ujohn /Paccounts
```

Command Switches

Figure 2.1 lists the Viewer command switches. The command switches identify the control table or file and provide log-on information that will give the user access to the database tables used in the reports.

Command switches can appear in any order on the command line and can be either upper or lower case. Each command switch must be attached to the item it specifies; spaces between the switch and the specification are not allowed. For example, to specify the control table password “admin” with the **/CP** switch, **`/CPadmin`** is correct and **`/CP admin`** is incorrect.

Switch	Description
/TS, /TT	Control table (TS) or text file (TT) name. Required unless you are running multiple reports with a text command file.
/CS	Data source for control table.
/CU, /CP	Control table user name (CU) and password (CP).
/CD	Database for control table.
/U, /P	User name and password for report tables.
/R	Default report directory.
/D	Default data directory.
/I	Default image file directory.
/O	Directory and/or name for output status file.
/H	Suppresses printing of Title and Summary lines when no records are found.
/B	Suppresses display of product "splash screen" at startup.
/AL	Name of DLL to be pre-loaded.

Figure 2.1 Command Switches

These switches are explained in detail in the following sections.

Control Table or File Name (/TS, /TT)

You must use the **/TS** or **/TT** switch with the Viewer command to specify the control table or file you are using. For example, to run all reports in a REPORTS control table (whose data source is "Sample Reports"), you would use this Viewer command:

```
RSWRUN /TSreports /CS"Sample Reports"
```

To run the report identified in the text control file REPORTS.TXT, you would use this command:

```
RSWRUN /TTreports
```

For control tables and files that are stored as DOS files, you can include a path with the file name. For example, the following command will run the report specified by the RUNIN.TXT file in the \DATA directory on the C: drive:

```
RSWRUN /TTc:\data\runin.txt
```

Control Table Data Source (/CS)

If you are using a control table, you must use the **/CS** switch to identify the data source for that table. For example, to run all reports in an Oracle REPORTS control table whose data source is PERSONNEL, you would use a command like this:

```
RSWRUN /TSreports /CSpersonnel
```

Control Table Database (/CD)

If the database platform for your control table supports multiple databases, you can use the /CD switch to specify or override the control table database. For example, to run all reports in a SQL Server REPORTS table in a database BENEFITS with a data source of PERSONNEL, you would use a command like this:

```
RSWRUN /TSreports /CSpersonnel /CDbenefits
```

Control Table User Name (/CU) and Password (/CP)

If your control table requires a different user name and password from your report tables, or if you do not use the **/P** and **/U** switches, you must use the **/CU** and **/CP** switches to supply the control table user name and password (if required for access to the control table). For example, if the user name “jane” and password “system” are required to access the Oracle control table INVOICE, you would use a command like the following:

```
RSWRUN /TSinvoice /CSsales /CUjane /CPsystem
```

If your database requires a user name but no password for access to the control table, you must still use the **/CP** switch if you want to bypass the log-on dialog; for example:

```
RSWRUN /TSinvoice /CSsales /CUmary /CP
```

User Name and Password for Report Tables (/U, /P)

If your database requires a user name and password for access, you can include the user name and password in the Viewer command. For example, if the user name "john" and password "accounts" are required to access the database used in the report specified in an Oracle INVOICE control table, you would use a command like the following:

```
RSWRUN /TSinvoice /CSsales /Ujohn /Paccounts
```

If you do not supply user name and password using /U and /P, the Viewer will try to access the database using information saved in the report and in the RSW.INI file, if one is present. For security reasons, passwords are not saved with reports or stored in RSW.INI. If the required access information is not available, the Viewer will prompt the user for log-on information. If your database requires a user name but no password for access, you must still include the /P switch if you want to bypass the log-on dialog; for example:

```
RSWRUN /TSinvoice /CSsales /Umary /P
```

Note that these switches are used to establish a connection to the database, as distinct from providing access to individually password-protected tables (as with Paradox, for example). For information about supplying passwords for access to individually password-protected tables, see the section in this chapter on the RI_DSOURCE parameter.

Default Report Directory (/R)

To specify a default directory where the Viewer may look for the report specified in the control file, use the /R switch in the command you use to call the Viewer. For example, the following command specifies C:\LIB as the default report directory:

```
RSWRUN /TTreports.txt /Rc:\lib
```

This command will run the report(s) specified in the REPORTS.TXT control file. The Viewer will look for the report(s) in C:\LIB. The default report directory you specify with this switch will override any default report directory specified in the RSW.INI file.

Default Data Directory (/D)

The Viewer looks for dBASE and Btrieve tables, indexes, and text memo files in the directories saved with the report. To specify a default directory where the Viewer will look for these data files when they are

not in the saved directory, use the **/D** switch. For example, the following command specifies C:\DATA as the default data directory:

```
RSWRUN /TTreports /DC:\DATA
```

The default data directory you specify with the /D switch will override any default data directory specified in the RSW.INI file.

Default Image File Directory (/I)

To specify a default directory where the Viewer may look for image files used in a 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 RSW.INI file.

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 selects MYSTATUS as the name for the status file generated by this Viewer command. Because no path is specified, the file will be created in the current directory.

```
RSWRUN /TTreports.in /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 file named RSWRUN.OUT in the specified directory. If you do not use the **/O** switch, the Viewer creates a status file named RSWRUN.OUT in the current directory.

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 RSW.INI for Default Information

ARPEGGIO Report Designer stores log-on information in the file RSW.INI, which is created in the Windows directory at installation. When you start Report Designer, the log-on information (except the password) is saved in RSW.INI. The default log-on information for each database platform is replaced whenever you connect to that platform. The Viewer will look for an RSW.INI file in the Windows directory and attempt to use the defaults if the Viewer command does not supply the log-on information. Any information you supply using Viewer command switches will always override the corresponding RSW.INI setting.

If you distribute reports to other users, you can customize RSW.INI for each user and distribute it with the other Viewer files. However, the command switches and control table parameters provide a more reliable way to supply accurate and up-to-date log-on information. Figure 2.2 lists the RSW.INI settings that the Viewer will use (unless a command-line switch is used instead).

RSW.INI Setting	*Specifies*
[Defaults] Section	
DataDir	Default data directory
ImgDir	Default image file directory
ImgExt	Default image file extension
LibDir	Default library file directory
MemExt	Default memo file extension
IndExt	Default index file extension
[Preferences] Section	
PrevWinClr	Preview window color
ShowSplash	Suppresses display of product “splash screen” at startup

Figure 2.2 RSW.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

To prepare 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 Viewer would output the report 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 2.10. For character columns, use any supported character data type. For numeric fields, 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, if running 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. When you use a text file to control the Viewer, you must create a separate 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 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 and 2.10; user-defined parameters can have any name.
- The only required parameters 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 contains predefined parameters specifying values that control frequently changed report features such as filters. 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 listed in Figures 2.3 and 2.10.

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

Width of Predefined Parameters

Predefined parameters have maximum widths specified in Figures 2.3 and 2.10. While you should not exceed these widths, you can decrease the widths of these parameters to correspond to the actual width of your data. 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 filter. By default ARPEGGIO is case insensitive, but if you edited RSW.SRT to make ARPEGGIO case sensitive, you should enter filter 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, and RI_INCLUDE.

Parameters for Modifying Report Characteristics

Figure 2.3 lists the predefined parameters in the Viewer control table or file 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 report execution.

Each parameter name has the prefix RI_. In the Data Type column, **C** represents the character data type, **N** represents numeric, and **L** represents logical.

Field Name	Contents	Data Type	Max. Width
RI_ALIAS1 – RI_ALIAS99	In each, a related table name	C	150
RI_BEGPAGE	Beginning page number	N	9
RI_CHKTIME	Checkpoint frequency flag	C	1
RI_COPIES	Number of copies	N	9
RI_DB	Database for report tables	C	50
RI_DISPERR	Display error flag	C or L	1
RI_DSOURCE	ODBC data source	C	32
RI_ENDPAGE	Ending page number	N	9
RI_EXPDST	Destination (display, file, or printer) for Excel exports	C	8
RI_FILTER	Filter expression	C	1024
RI_GROUP1 – RI_GROUP8	Group field override	C	50
RI_ID	Row identifier (control tables only)	N	4
RI_INCLUDE	Filter flag	C	1
RI_LIBRARY	Report library name	C	128
RI_MASTER	Master table name	C	128
RI_MEMO	Text memo file name	C	128
RI_NOESC	User escape flag	C or L	1
RI_OUTFILE	Output file name	C	128
RI_PRINTER	Destination	C	32
RI_REPLACE	SQL SELECT, EXEC, or DEFINE REPORTVIEW overrides	C	1024
RI_REPORT	Report name	C	30
RI_REPPICK	? or R to prompt for report	C	1
RI_SORT1 – RI_SORT8	Sort field override	C	51
RI_STATUS	Display status flag	C or L	1
RI_TEST	Test pattern flag	C or L	1
RI_WHERE	SQL WHERE clause	C	1024
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

The following sections, arranged in alphabetical order by parameter name, explain the values required by each parameter.

Parameters that require character values can contain upper, lower, or mixed case letters. Note that some database platforms can be configured for case sensitivity; if your database platform is case sensitive, make sure to enter database, table, and column names in the proper case.

RI_ALIAS1 – RI_ALIAS99

These parameters are optional. You can use each of the parameters to specify a related table to override those saved with the report. The syntax is:

```
<alias> = <table name>
```

In this specification, <alias> represents the R&R alias assigned to the table in the saved report and <table name> represents the replacement table.

For example, the following specification replaces the related table assigned the CUST95 alias in the saved report with a table named CUST96:

```
CUST95 = CUST96
```

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 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. (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 determines how often the Viewer status file, by default RSWRUN.OUT, is updated. The checkpoint flag can contain the letter **R** or **P**. **R** tells the Viewer to update the status file after completing each report; **P** tells the Viewer to update the file 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 file after each page of the report is processed. (See the section in this chapter 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_DB

This parameter is intended for use with SQL Server data sources only. Use it to override the database for the master table and any related table whose database matches that of the master. Note that this parameter will not override any qualified table name specified in a User-SQL report.

RI_DB is ignored if a value is supplied for RI_MASTER.

RI_DISPERR

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

RI_DSOURCE

Use this parameter to specify (or override) the ODBC data source for the report tables. You can include an optional connect string addendum that will be appended to the string used to connect to a data source. Syntax is as follows (separator is three vertical bars):

```
<data source name> ||| <connect string addendum>
```

RI_DSOURCE can also be used to supply one or more passwords for access to individual tables if the database driver supports passwords in the connect string (as the Paradox driver does, for example). See the on-line help for the driver for information about connect string options.

RI_EXPDST

Use this parameter to specify the destination (display, file, or printer) for a report that has been 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 specifies a logical expression that will override the filter saved with a report, if any, when the value in RI_INCLUDE is **O** for Override.

The syntax of the RI_FILTER expression is identical to that of a calculated field expression that returns a logical value. The RI_FILTER expression can be up to 1024 characters long. When an expression is specified, 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 filter 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 filter expression **LIKE("*Dallas*",NOTE)** 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 greater than or equal to 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 filter expressions can be entered by using parentheses. For example, the following filter 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 RI_INCLUDE *must* be **O** in order for the RI_FILTER override to take effect. If you omit RI_INCLUDE, the RI_FILTER value will be ignored and the report will be run using the saved filter (if any).

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_ID

This parameter is for database control tables only. The RI_ID parameter allows you to assign a job number to each report in your control table. You can then use one or more RI_ID values with the Viewer command to specify which reports you want to run.

RI_ID values need not be unique. You can assign the same number to multiple reports, creating a set. You can then run the set by entering a single RI_ID number on the Viewer command line.

RI_INCLUDE

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

- **S** (Saved) means to run the report using the filter saved with it, if any. The expression in RI_FILTER will be ignored and the report will be run exactly as it was saved.
- **E** (Entire) means to run the entire report, ignoring any filter saved in the report or contained in the RI_FILTER parameter.
- **O** (Override) means to override the saved filter with the expression in the RI_FILTER parameter. The report will be run with the records selected by the RI_FILTER expression.
- **?** (Question mark) means to display a dialog box allowing the user to enter a filter expression or edit the filter saved with the report. If no filter was saved with the report, the Insert Condition dialog will display, as shown in Figure 2.5.

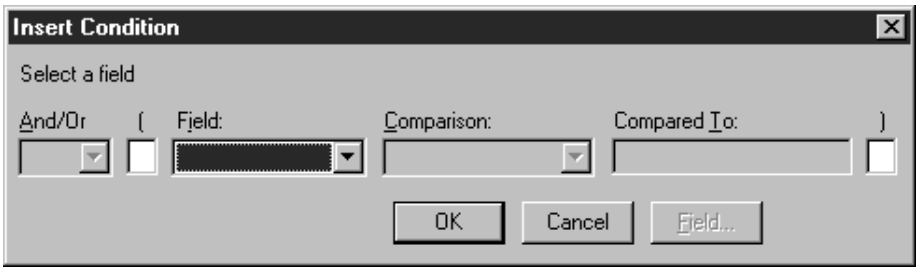

Figure 2.5 Insert Condition Dialog Box

If a filter was saved with the report, the Filter dialog box will display, as shown in Figure 2.6.

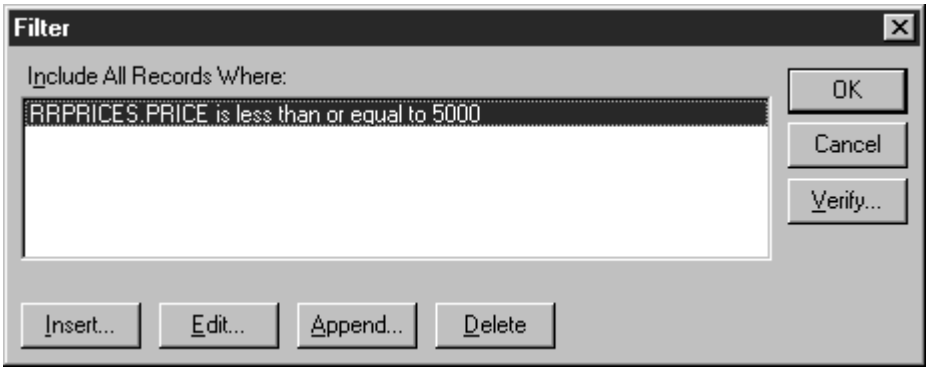

Figure 2.6 Filter Dialog Box

When you use the question mark (**?**) value for the RI_INCLUDE parameter, the value of RI_FILTER is always ignored.

Note that RI_INCLUDE has no impact on the RI_WHERE parameter. If RI_WHERE is specified, it will always be evaluated by your SQL software directly; any filter will be applied to the result.

RI_LIBRARY

This parameter is necessary only if you are running reports from an R&R Report Writer, SQL Edition, report library file; it identifies the library containing the report(s) you want to run. The library name can include a path. The .RP6 extension is optional. For example, a value of C:\PROJECT\CUSTOMER identifies the report library as CUSTOMER.RP6 in the subdirectory \PROJECT on drive C.

If you don't include a path, the Viewer searches for the file in the default report directory specified with the /R switch on the command line. If no default is specified on the command line, the Viewer searches for the library in the default directory specified in the RSW.INI file. If RSW.INI is not present and no default report directory is specified, the Viewer searches for the library in the current directory.

If you leave this parameter blank or if the library you specify cannot be found or read, the Viewer writes an error in the status file and, optionally, displays an error message box (see RI_DISPERR).

RI_MASTER

This parameter is optional. It contains the name of a table that will override the master table saved with the report. The master table you specify should have the same columns as the master table originally used in the report.

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 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 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 as specified in RSW.INI or as overridden on the command line.

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

RI_NOESC

This parameter is optional. The user escape flag can be either true (**T**) or false (**F**). 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 status window appears only when RI_STATUS is set to true). The default value is false. Note that pressing Cancel will *not* interrupt execution of the Viewer during processing of a SELECT statement by a server.

If the user cancels the report, the RO_ECODE entry in the status file contains a C (see the section in this chapter entitled **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 output file name 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 RI_OUTFILE:

```
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 or printer) at report execution. When the value of RI_PRINTER is a question mark, the user will see the dialog box shown in Figure 2.7. 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.

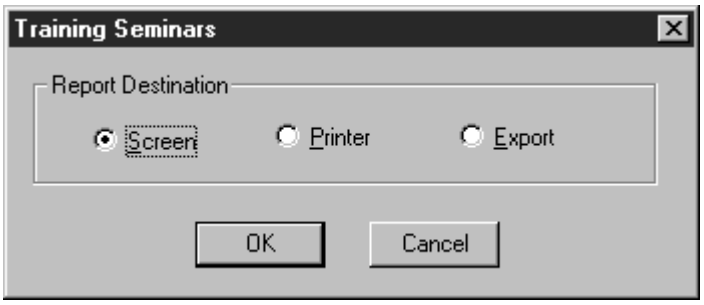

Figure 2.7 Print Destination Dialog Box

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.

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_REPLACE

The optional RI_REPLACE parameter allows you to supply a substitute value to override all or part of the SELECT, EXEC, or DEFINE REPORTVIEW statement used to select rows for a User-SQL report.

When you enter a SELECT, EXEC, or DEFINE REPORTVIEW statement in Report Designer, you must enclose in double angle brackets (<< >>) any portion that you may want to replace at report execution. Using RI_REPLACE, you can provide substitute values for the delimited portions, leave them intact, or specify that you want them to be ignored at report execution. You can delimit any text in the statement except the initial commands SELECT, EXEC, and DEFINE REPORTVIEW, which cannot be substituted. The initial SELECT, EXEC, or DEFINE REPORTVIEW must be followed by a space. Note also that nesting parameters is not allowed — do not insert delimiters within delimiters.

The syntax of RI_REPLACE is a comma-separated list of parameters enclosed in double angle brackets:

```
<<param1>>,<<param2>>,<<param3>>,...<<paramN>>
```

The number of parameters in the RI_REPLACE value *must match exactly* the number of delimited portions of the SELECT, EXEC, or DEFINE REPORTVIEW statement saved with the report. Everything between delimiters will be substituted exactly as entered in place of the corresponding delimited text in the original statement. Space outside delimiters is ignored.

For example, suppose you are using this SELECT statement:

```
SELECT *
FROM customers
WHERE state='MA'
ORDER BY last_name
```

You can delimit any parts of the statement except the initial word SELECT. For example, you might delimit the FROM, WHERE, and ORDER BY clauses, as shown in this example:

```
SELECT *
<<FROM customers>>
<<WHERE state='MA'>>
<<ORDER BY cust_name>>
```

To provide substitutions for the three delimited sections of the SELECT statement, you might supply the following RI_REPLACE value in your control table:

```
<<FROM customers,sales>>,
<<WHERE customers.cust_no=sales.cust_no AND state='CA'>>,
<<ORDER BY sale_date>>
```

To leave any delimited portion intact, use a comma as a place holder. To replace the WHERE clause and leave the FROM and ORDER BY clauses intact, you might use this RI_REPLACE value:

```
,<<WHERE state='CA'>>,
```

When you do not want a delimited portion of the statement to be applied, use empty delimiters (<< >>) to specify a null replacement value. For example, this RI_REPLACE value specifies that the FROM clause of the original SELECT should be left intact, and the WHERE and ORDER BY clauses should be ignored:

```
,<<>>,<<>>
```

In general, the application of the RI_REPLACE parameter must yield a SELECT statement that would itself be valid as the basis of a User-SQL

report. For example, all columns in the result of the modified SELECT must be uniquely named. In addition, any columns returned by the original User-SQL SELECT that are used in the report must also be returned by the modified SELECT with the same names and types.

Note that RI_REPLACE values are not applied to Auto-SQL reports (reports created by selecting master and related tables). To insert a WHERE clause in the SQL statement for an Auto-SQL report, use the RI_WHERE parameter.

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," enter **Order Invoice** as the value for this parameter (note that you do not need to include the file 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 file 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 field; 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_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 2.8 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 2.8 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 is true (**T**), the Viewer will display a Status window. If RI_NOESC is set to false, the Status window will contain a Cancel choice that allows the user to terminate a report in progress. Note that pressing Cancel will *not* interrupt execution of the Viewer during processing of a SELECT statement by a server.

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

RI_TEST

This parameter is optional. The test pattern flag can be either true (**T**) or false (**F**). 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. If the parameter is blank, the user is not offered the choice of printing 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_WHERE

The optional RI_WHERE parameter enables the Viewer to insert a WHERE clause in the SQL statement for an Auto-SQL report. If you or your users are proficient in SQL, you may want to use this parameter instead of RI_FILTER and RI_INCLUDE to select records. Since the WHERE clause is evaluated directly by the SQL software, using RI_WHERE can improve performance and enable you to make use of any WHERE clause supported by your SQL software.

The WHERE clause specified with this parameter always affects the report, regardless of whether a filter was saved with the report. If you have also used RI_FILTER and RI_INCLUDE to select records, the effect of RI_WHERE is as follows:

- If RI_INCLUDE is **S** for "Saved," both the filter saved with the report *and* the clause in RI_WHERE are used to select records.
- If RI_INCLUDE is **O** for "Override," both the filter expression in RI_FILTER *and* the clause in RI_WHERE are used to select records.
- If RI_INCLUDE is **E** for "Entire," *only* the RI_WHERE clause is used to select records.
- If RI_INCLUDE is a question mark (**?**) to allow the user to enter a filter interactively, both the user's filter expression and the RI_WHERE clause are used to select records.

Note that RI_WHERE values are not applied to User-SQL reports. To override the selection conditions for a User-SQL report, use the RI_REPLACE parameter.

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 2.9. 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 (that is, you can enter it in upper, lower, or mixed case). If you enter a value for this parameter and RI_WPORT is blank, Viewer uses the port associated with the printer name in the list of available printers.
- The question mark (**?**) value, to allow the user to select a printer at report execution. When RI_WPTR contains a question mark, the Print dialog displays, as shown in Figure 2.9.

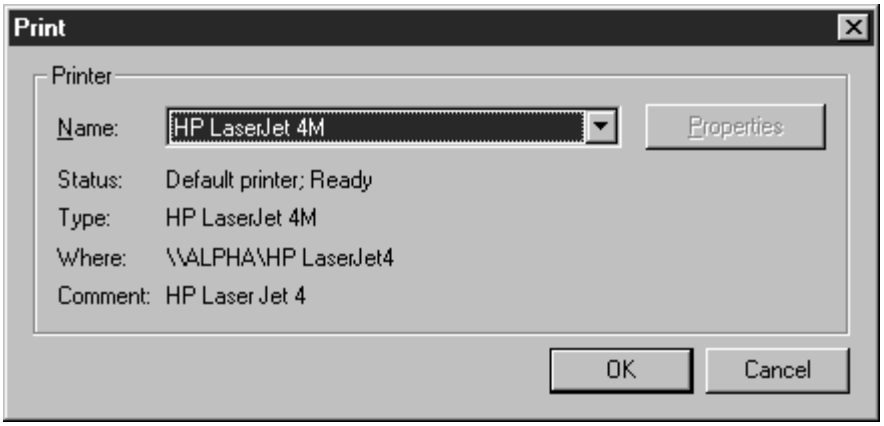

Figure 2.9 Print Dialog Box

- 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)

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.

If this parameter is blank, the printer saved with the report will be used. 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") to be displayed in the following:

- 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, or any user-defined parameter value.

If this parameter is blank, the report name will be used for the title.

Parameters to Control Viewer Preview Display

The parameters listed in Figure 2.10 control the size and appearance of the preview window at report execution.

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 2.10 Control Parameters for Report Preview

These parameters are explained in the following sections.

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.

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 report execution 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 report execution. 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 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 description of the Viewer Control Parameters in the **Using Control Tables and Files** section of this chapter.

By default, the Viewer creates a status file called RSWRUN.OUT in the current directory. You can specify a different file name or alternate directory using the **/O** switch. The status file is in Windows .INI-style format and has the header [RSW Runtime output].

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.

The Viewer will always create a status file unless one of the following conditions prevents it:

- A Viewer command-line error;
- A link directory error;
- A disk error;
- Insufficient memory.

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 the **/O** switch to specify a unique status file for each user.

Status File Entries

Figure 2.11 lists the entries in the status file. Each entry name has the prefix RO_.

Entry Name	*Contents*
RO_ECODE	Error code
RO_EMSG	Error message
RO_REPORTS	Number of reports completed
RO_PAGES	Last report page completed
RO_RIRECNO	RI_ID value of last control table record processed

Figure 2.11 Entries in Viewer Status File (RSWRUN.OUT)

Entries appear one per line in the format <entry> = <value>, as in RO_ECODE = N.

RO_ECODE

The error code entry contains one of the following characters:

- **N** – The Viewer completed without error; the RO_EMSG value is blank.
- **C** – The user selected Cancel to cancel a report; the RO_EMSG message is "Report canceled."
- **J** – There is a syntax error in the Viewer command or the control file; see the message in the RO_EMSG entry.
- **R** – There is an error caused by the report definition or by the value in a control file parameter; see the RO_EMSG entry.

RO_EMSG

The error message value is blank if Viewer processing completed without an error. If processing was canceled for any reason, this entry 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 displayed when you attempt to output the report from within Report Designer.

If RO_ECODE contains **J**, there is an error in the Viewer command or in the control table or file.

RO_REPORTS

This entry contains the number of reports that were completed. For example, if the Viewer is called to print three reports and the printer jams during the second report, this entry contains the number 1, indicating that one report was completed. Use this number to determine which report(s) did not complete.

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 is executed with a multi-report command file or control table). 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 file is **P**. See the section in this chapter entitled **Using Control Tables and Files** for a description of RI_CHKTIME.

RO_RIRECNO

If you are controlling the Viewer with a control table, this field contains the RI_ID value of the record specifying the last report that was processed. This number uniquely identifies the report that caused the error if you used unique RI_ID values for each row in the table.

Application Calls to the Report Viewer

This section provides examples illustrating how you can incorporate calls to the Viewer in C, Visual Basic, and Power Builder applications. Note that these examples are provided only to demonstrate the syntax for calls to the Viewer for each language.

If your application includes code to check the Viewer status file (by default, RSWRUN.OUT), keep in mind that up-to-date information from that file will not be available until Viewer reports complete. To avoid confusion, delete any existing status file before calling 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 2.12.

```
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,"rswrun /TT%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 2.12 Calling the Viewer from a C Program

In this example, the call to the Viewer includes the /TT switch to identify the text control file and the /R switch to specify a default report directory. You can include any combination of the command switches explained in the **Viewer Command Switches** section. Note that the second parameter supplied to WinExec, SW_SHOW, is ignored if you have included control table values to govern the 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 control table values. Figure 2.13 illustrates a subroutine that could be used to call the Viewer from a Visual Basic application.

```
Sub Command1_Click ()
cmd$ = "c:\rsw\rswrun.exe /TTc:\rsw\rrsample\rswrunin.txt"
i% = Shell(cmd$, 1)
End Sub
```

Figure 2.13 Sample Visual Basic Subroutine Calling the Viewer

In this example, Viewer is executed using the Viewer control file in C:\RSW\RRSAMPLE.

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 2.14 illustrates a script that could be used to call the Viewer from a PowerBuilder application.

```
SetProfileString("D:\RSWRUN.IN","rswrun","ri_copies","2")
run("RSWRUN.EXE /TTd:\rswrun.in /Od:\rswrun.out /U" +
   sqlca.logid + "/P" + sqlca.logpass)
```

Figure 2.14 Sample PowerBuilder Script Calling the Viewer

Chapter 3
Parameter Passing

Introduction

You can control some features of the layout and content of reports 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 report execution. You might want to allow the user to select the sort order for the report. Rather than creating several different reports, you can create one report and present the user with a menu that offers a choice of sort options. You then pass the user's choice to the Viewer and use a calculated field expression to determine the sort field.

You can pass parameters to a report in two ways:

- Define special parameters in the Viewer control table or file. At report execution, prompt the user to enter values and 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, join 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 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 more detail.

Defining Parameters

In addition to the predefined parameters listed in Chapter 2, "Using the Report Viewer," your control table or file can include parameters you define. A parameter can have any name you like (for example CONAME), and it 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 that takes a character string. 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 question mark can be followed by 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:".

As a result, the user will see a dialog like the one in Figure 3.1.

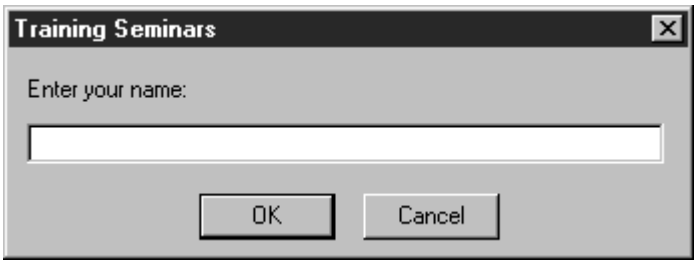

Figure 3.1 Viewer Dialog Box with Prompt

The size and shape of this dialog box is the same for all user-defined parameters. The title bar will contain the value of the RI_WTITLE parameter. If RI_WTITLE is not specified, the Viewer uses the report name. The Viewer prohibits the user from entering more than 512 characters. If the user selects the Cancel button, the report will not run and the "Canceled" message will be written to the status file.

If your control table parameter contains a **?** without a text string, the Viewer displays the dialog box shown in Figure 3.1 with the prompt "Enter value for (PARAMETER NAME)", as in "Enter value for AUTHOR".

Incorporating User-Supplied Values in Reports

Once the user has entered or selected a value, you pass the value to your report using the RIPARAM( ) function in a calculated field expression. RIPARAM( ) takes a control file parameter as its argument and returns the value of the parameter as a string. In this way, the user's input becomes available to the report.

For example, in a general ledger application, you might create a control file 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 R&R 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.

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 joining 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 database software.
2. In Report Designer, create a new report or retrieve and modify an existing report.
3. Create a calculated field that you will use to join the master table to the parameter table.
4. Join the master table to the parameter table using the R&R calculated field as the join field from the master table and a unique ID field in the parameter table to match the calculated join field in the master table.
5. Use the fields from the parameter table in your report and save the report.
6. Create a database program that will get the information from the user, store it in the parameter table, and call the Viewer.

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

Creating the Parameter Table

Your 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 column for each parameter. You can use any data type. Assign an appropriate column width.

The parameter table must contain a column that serves as a unique identifier for each row in the table. You will use this column to join the parameter table to the master table in Report Designer. The column can have any data type supported by Report Designer.

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

the table should have a row for each report. In this case, you might use the report name as the join column in the parameter table.

Creating the Calculated Join Field

In Report Designer, use Calculations ⇒ Calculated Field to create a calculated field to join the master table to the parameter table. The calculated field's expression should be a constant equal to the join column value for the appropriate parameter table row. If your parameter table has a single row, you might have used the number 1 as the join column value. In that case, the calculated field's expression is simply 1. If your parameter table has multiple rows, you might have used the report name as the join column. In that case, the calculated field's expression is the report name that identifies the appropriate row, as in "INVOICE1."

The calculated join field's expression must be database-evaluatable. Report Designer precedes the names of database-evaluatable calculated fields with an X in field lists.

Joining the Master Table and the Parameter Table

Use Database ⇒ Joins to join the master table and the parameter table. Use the calculated field as the join field from the master table and the ID column as the join field from the parameter table.

Chapter 4
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. The other methods are explained in Chapter 2, "Using the Report Viewer," and Chapter 5, "Using the ARPEGGIO Custom Control."

Two sample applications demonstrating use of the Viewer DLL are installed into appropriate subdirectories of the SAMPLE directory:

- SAMPC, which is installed in SAMPLE\C, demonstrates the use of the DLL in C code.
- SAMPMFC, which is installed in SAMPLE\MFCDLL, demonstrates the use of the DLL in MFC code.

Each sample has an accompanying README file that explains it in more detail.

The Viewer DLL provides a direct application programming interface to the ARPEGGIO Viewer. The general logic of using this API to invoke the Viewer is as follows:

- ❑ 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 Viewer control parameters.
- ❑ Use **writeRuntimeRecord** to save the parameters in a Viewer Control File for later execution or **execRuntime** to use the ARPEGGIO 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 Viewer DLL or a specific report, to run a report, and to free resources used in working with a report or the Viewer 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 report or 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 Viewer 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.
- **getDataSource** gets the value of the data-source 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.
- **getFirstJoinInfo** gets the values of parameters pertaining to the first related table used in the report.
- **getFirstReplace** gets the value of the first replaceable string in the User-SQL statement saved with the report.
- **getFirstSortField** gets the name of the report's first sort field.
- **getFirstUserParam** gets the name of the first user-parameter used in the report.

- **getLibrary** gets the name of the report library 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.
- **getNextJoinInfo** gets the values of parameters pertaining to the next relation used in the report.
- **getNextReplace** gets the value of the next replaceable string in the User-SQL statement saved with 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.
- **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.
- **setDatabase** sets the value of the database parameter.
- **setDataDir** specifies an override for the default data directory.
- **setDataSource** specifies a data source.
- **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.
- **setImageDir** specifies an override for the default image directory.
- **setJoinInfo** sets the values of parameters pertaining to a related table used in the report.
- **setLibrary** specifies a report-library.
- **setLibraryDir** specifies an override for the default library directory.
- **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.
- **setPassword** specifies the password for logging into a database.
- **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 printer port.
- **setReplace** specifies a User-SQL replacement string.
- **setReportPick** specifies the optional use of a report-selection dialog in the Viewer that allows the user to select one or more reports at report execution.

- **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.
- **setUserName** specifies the user name for logging into a database.
- **setUserParam** specifies a value for a user-parameter used in the report.
- **setWhere** sets the value of an additional or replacement where clause for Auto-SQL reports.
- **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.

User-Interface Routines

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

- **chooseDataSource** is used to present the user with a dialog from which to select a data source.
- **chooseTable** is used to present the user with a dialog from which to select a table.

- **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 make the Viewer DLL forget 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.

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 RSRRPT32.H, for use in C/C++ programs, and one named RSDECL32.BAS for use in Visual Basic programs.

chooseDataSource

BOOL FAR PASCAL chooseDataSource (int *hReport*, **LPSTR** *lpszDataSource*, **int** *dsSize*);

hReport	Report handle.
lpszDataSource	Address of buffer in which to return selected data source and in which to optionally specify an initial data source.
dsSize	Size of *lpszDataSource* buffer.

Return Value

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

Description

Use **chooseDataSource** to allow the user to interactively select a new data source for use with the report specified by *hReport*. The name of the data source selected by the user will be returned in the buffer specified by *lpszDataSource* to the extent allowed by *dsSize*. If, on entry, *lpszDataSource* contains the name of an available data source, that data source will be highlighted initially in the list of data sources presented to the user. The value of the data source selected by the user will become the current data source. There is no need to call **setDataSource** to make it current.

Related Functions

getDataSource, **setDataSource**

Example

To allow the user to select a new data source for the report whose handle is *hRpt*, showing the current data source as the initial choice:

```
{
   char buf[100];

   getDataSource (hRpt, (LPSTR)buf, 100);
   chooseDataSource (hRpt, (LPSTR)buf, 100);
}
```

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 *lpszPrinter* buffer.
lpszPort	Address of buffer in which to return selected printer port.
poSize	Size of *lpszPort* buffer.

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*, **LPSTR** *lpszUserName*, **LPSTR** *lpszPassword*, **LPSTR** *lpszDataSource*, **int** *dSize*)**;**

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.
r*size*	Size of *lpszRepName* buffer.
lpszUserName	Name of user, for connecting to data source.
lpszPassword	Password, for connecting to data source.
lpszDataSource	Name of data source, or buffer in which to return name of data source.
dSize	Size of *lpszDataSource* 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**.

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 rSize (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.

The *lpszUserName* and *lpszPassword* arguments are optional and can be used to specify a username and password, respectively for use when connecting to the data source to be used with the report specified by *lpszRepName* or chosen by the user.

The *lpszDataSource* argument is optional and can be used to specify a data source override for the specified report. This can be useful if the specified report either contains no data source information, or if the data source in use when the report was saved no longer exists or does not exist on the current system. The *lpszDataSource* argument can be:

- the name of a data source, in which case the data source saved with the report (if any) is ignored in favor of the specified data source.
- a pointer to an empty buffer of size *dSize*, in which case the data source saved with the report will be used, if possible, or if the saved report has no data source or its data source is invalid, the user will be presented with a

"choose data source" dialog. The data source's name will be returned in *lpszDataSource* to the extent allowed by *dSize*.

- NULL, in which case the data source saved with the report will be used, if possible, or if not possible one of the errors: "No data source specified in report" or "Cannot find data source 'name'" will be returned.

Related Functions

endReport, getRuntimeRecord, setUserName, setPassword

Example

To use the report "Guests" from the library c:\libs\reports.rp6 with a username of "Jack Paar", a password of "Tonight", and a data source of "Entertainers":

```
{
   int hRpt;
   hRpt = chooseReport ((LPSTR)"Application Name",
         (LPSTR)"c:\\libs\\reports.rp6", (LPSTR)"Guests",
         (LPSTR)"Jack Paar", (LPSTR)"Tonight",
         (LPSTR)"Entertainers", 13);
}
```

chooseTable

BOOL FAR PASCAL chooseTable (int *hReport*, **LPSTR** *lpszTable*, **int** *tSize*, **LPSTR** *lpszDataSource*, **int** *dsSize*, **LPSTR** *lpszDatabase*, **int** *dbSize*);

hReport	Report handle.
lpszTable	Address of buffer in which to return table name.
tSize	Size of buffer pointed to by *lpszTable*.
lpszDataSource	Address of buffer in which to return data source name or in which a data source is specified.
dsSize	Size of buffer pointed to by *lpszDataSource*.
lpszDatabase	Address of buffer in which to return database name or in which a database is specified.
dbSize	Size of buffer pointed to by *lpszDatabase*.

Return Value

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

Description

Use **chooseTable** to allow the user to interactively select a new table, database, and data source for use with the report specified by *hReport*. The name of the table selected by the user will be returned in the buffer specified by *lpszTable* to the extent allowed by *tSize*.

If, on entry, *lpszDataSource* contains the name of an available data source, **chooseTable** will use that data source in determining the list of tables presented to the user. If there is no current data source or *lpszDataSource* does not specify an available data source, the user

will be allowed to select a data source and **chooseTable** will return the selected data source name in *lpszDataSource* to the extent allowed by *dsSize*.

If, on entry, *lpszDatabase* contains the name of a valid database (for those platforms that support multiple databases), **chooseTable** will initially select that database in displaying the list of tables. **chooseTable** will return the name of the database from which the user selects a table in *lpszDatabase* to the extent allowed by *dbSize*.

Related Functions

chooseDataSource, **getDataSource**, **setDataSource, getMasterTableName, setMasterTableName**

Example

To allow the user to select a new data source and table for the report whose handle is *hRpt*, and then make the selected data source current and use the selected table to replace the master table :

```
{
   char dsbuf[100];
   char dbbuf[100];
   char tbuf[100];
   dsbuf[0] = 0;
   dbbuf[0] = 0;
   chooseTable (hRpt, (LPSTR)tbuf, 100, (LPSTR)dsbuf, 100,
                (LPSTR)dbbuf, 100);
   setDataSource (hRpt, (LPSTR)dsbuf);
   setMasterTableName (hRpt, (LPSTR)tbuf);
}
```

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 RSWRUN 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 condition resulting from the use of **execRuntime** is available through **getErrorInfo**. In particular, in the event of a **WinExec** error, **getErrorInfo** returns a message containing both the **WinExec** error code and descriptive text.

If you have used **setStatusEveryPage** to request that the Viewer status be updated after every page, *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 Viewer status file and the information provided by *lpiECode*, *lpiPageCount*, and *lpiEMsg* are instead provided by the fields RO_ECODE, RO_PAGES, and RO_EMSG. See Chapter 2 for details of the Viewer 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);
}
```

getDataSource

BOOL FAR PASCAL getDataSource (int *hReport*, **LPSTR** *lpszDataSource*, **int** *dsSize*);

hReport	Report handle.
lpszDataSource	Data-source buffer.
dsSize	Size of *lpszDataSource* buffer.

Return Value

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

Description

Use **getDataSource** to obtain the data source currently associated with the report specified by *hReport*. **getDataSource** returns the data source name in the buffer pointed to by *lpszDataSource* to the extent allowed by *dsSize*. See **setDataSource** for a discussion of this parameter.

Related Functions

setDataSource, **chooseDataSource**

Example

To allow the user to select a new data source for the report whose handle is *hRpt*, showing the current data source as the initial choice:

```
{
   char buf[100];
   getDataSource (hRpt, (LPSTR)buf, 100);
   chooseDataSource (hRpt, (LPSTR)buf, 100);
}
```

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 second 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 Viewer 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 following example.

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 ((LPSTR)emsg, 200, (LPINT)&ecode);
   // ecode will have an error code
   // emsg will have an error message, truncated to 200
   //  bytes, if necessary
   // your 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 ARPEGGIO 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 ARPEGGIO 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

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 and 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);
   }
```

getFirstJoinInfo

BOOL FAR PASCAL getFirstJoinInfo (int *hReport*, **LPSTR** *lpszTable*, **int** *tSize*, **LPSTR** *lpszAlias*, **int** *aSize*);

hReport	Report handle.
lpszTable	Related-filename buffer.
tSize	Size of *lpszTable* buffer.
lpszAlias	Alias buffer.
aSize	Size of *lpszAlias* buffer.

Return Value

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

Description

Use **getFirstJoinInfo** to obtain information about the "first" related table in the report specified by *hReport*. **getFirstJoinInfo** returns the related table's name in the buffer pointed to by *lpszTable* to the extent allowed by *tSize* and the alias of the related table in the buffer pointed to by *lpszAlias* to the extent allowed by *aSize*. Use **getNextJoinInfo** 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

getNextJoinInfo, **setJoinInfo**, **getMasterTable**

Example

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

```
{
   char table[260];
   char alias[10];

   getFirstJoinInfo (hRpt, (LPSTR)table, 260, (LPSTR)alias, 10);
}
```

getFirstReplace

BOOL FAR PASCAL getFirstReplace (int *hReport*, **LPSTR** *lpszReplace*, **int** *size*);

hReport	Report handle.
lpszReplace	Replacement-string buffer.
size	Size of *lpszReplace* buffer.

Return Value

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

Description

Use **getFirstReplace** to get the first replaceable portion of the SQL statement associated with the User-SQL report specified by *hReport*. **getFirstReplace** returns the first replaceable portion in the buffer pointed to by *lpszReplace* to the extent allowed by *size*. Use **getNextReplace** in a loop to get the remaining replaceable portions of the SQL statement. The replaceable portions of the User-SQL statement are surrounded by pairs of angle brackets, as in

Select <<'firstname', 'lastname'>> from students
where <<'firstname' < 'lastname'>>

which has replaceable portions of **<<'firstname','lastname'>>** and **<<'firstname' < 'lastname'>>**. See **getErrorInfo** for information about how to detect end-of-list.

Related Functions

getNextReplace, **setReplace**, **setWhere**

Example

To get information about the first replaceable portion of the SQL statement from the User-SQL report whose handle is *hRpt*:

```
{
   char buf[500];

   getFirstReplace (hRpt, (LPSTR)buf, 500);
}
```

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);
}
```

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*:

```
{
   char lib[80];

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

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**.

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. See **setReportPick** for a discussion of selection of reports by the user.

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

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 of FILTER_ID_SORT to both.

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**.

getNextJoinInfo

BOOL FAR PASCAL getNextJoinInfo (int *hReport*, **LPSTR** *lpszTable*, **int** *tSize*, **LPSTR** *lpszAlias*, **int** *aSize*);

hReport	Report handle.
lpszTable	Related-table buffer.
tSize	Size of *lpszTable* buffer.
lpszAlias	Alias buffer.
aSize	Size of *lpszAlias* buffer.

Return Value

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

Description

Use **getNextJoinInfo** in a loop to obtain information about all related tables but the "first" in the report specified by *hReport*. **getNextJoinInfo** returns the related table's name in the buffer pointed to by *lpszTable* to the extent allowed by *tSize*, and the alias of the related table in the buffer pointed to by *lpszAlias* to the extent allowed by *aSize*. Use **getFirstJoinInfo** to obtain equivalent information about the "first" of the related tables. See **getErrorInfo** for information about how to detect end-of-list.

Related Functions

getFirstJoinInfo, **setJoinInfo**, **getMasterTable**

Example

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

```
{
   char table[260];
   char alias[10];

   getNextJoinInfo (hRpt, (LPSTR)table, 260, (LPSTR)alias, 10);
}
```

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

getNextReplace

BOOL FAR PASCAL getNextReplace (int *hReport*, **LPSTR** *lpszReplace*, **int** *size*);

hReport	Report handle.
lpszReplace	Replacement-string buffer.
size	Size of *lpszReplace* buffer.

Return Value

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

Description

Use **getNextReplace** in a loop to get all replaceable portions but the first of the SQL statement associated with the User-SQL report specified by *hReport*. **getNextReplace** returns the next replaceable portion in the buffer pointed to by *lpszReplace* to the extent allowed by *size*. Use **getFirstReplace** to get the first replaceable portion. The replaceable portions of the User-SQL statement are surrounded by pairs of angle brackets, as in

Select <<'firstname', 'lastname'>> from students
where <<'firstname' < 'lastname'>>

which has replaceable portions of <<'firstname','lastname'>> and <<'firstname' < 'lastname'>>. See **getErrorInfo** for information about how to detect end-of-list.

Related Functions

getFirstReplace, **setReplace**

Example

To get information about the next replaceable portion of the SQL statement from the User-SQL report whose handle is *hRpt*:

```
{
   char buf[500];
   getNextReplace (hRpt, (LPSTR)buf, 500);
}
```

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

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*. **getFirstSort Field** 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 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 as a single character in

the buffer specified by lpszDest to the exten 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 the report whose handle is *hRpt*:

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

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*, **LPSTR** *lpszUserName*, **LPSTR** *lpszPassword*, **LPSTR** *lpszDataSource*, **int** *dSize*) ;

lpszAppName	Name of calling application.
lpszControlFile	Pointer to ASCII Viewer control filename.
lpszUserName	Name of user, for connecting to data source.
lpszPassword	Password, for connecting to data source.
lpszDataSource	Name of data source or buffer in which to return data source name.
dSize	Size of lpszDataSource buffer.

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 *lpszUserName* and *lpszPassword* arguments are optional and can be used to specify a username and password, respectively, for use when connecting to the data source to be used with the report specified by *lpszRepName* or chosen by the user.

The *lpszDataSource* argument is optional and can be used to specify a data source override for the specified report. This can be useful if the specified report either contains no data source information, or if the data source in use when the report was saved no longer exists or does not exist on the current system. The *lpszDataSource* argument can be:

- The name of a data source, in which case the data source saved with the report (if any) is ignored in favor of the specified data source.
- A pointer to an empty buffer of size *dSize*, in which case the data source saved with the report will be used, if possible, or if the saved report has no data source

or its data source is invalid, the user will be presented with a "choose data source" dialog. The data source's name will be returned in *lpszDataSource* to the extent allowed by *dSize*.

- NULL, in which case the data source saved with the report will be used, if possible, or if not possible one of the errors: "No data source specified in report" or "Cannot find data source 'name'" will be returned.The handle returned by **getRuntimeRecord** is used as input to most other functions contained within this API.

Related Functions

chooseReport, getNewReportHandle, writeRuntimeRecord

Example

To use the report specified in the control file c:\libs\rrunin.txt with a username of "Jack Paar", a password of "Tonight", and a data source of "Entertainers":

```
{
   int hRpt;

   hRpt = getRuntimeRecord ((LPSTR)"Application Name",
      (LPSTR)"c:\\libs\\rrunin.txt", (LPSTR)"Jack Paar",
      (LPSTR)"Tonight", (LPSTR)"Entertainers", 13);
}
```

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 Viewer 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 ARPEGGIO 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);
```

setDatabase

BOOL FAR PASCAL setDatabase (int *hReport*, **LPSTR** *lpszDatabase*);

hReport	Report handle.
lpszDatabase	Default database.

Return Value

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

Description

Use **setDataDatabase** to replace the default database saved with the report specified by *hReport* with the value specified by *lpszDatabase*. ARPEGGIO may use the default database in trying to locate tables used in the report specified by *hReport*.

Related Functions

none

Example

To specify the use of accounting as the default database for the report whose handle is *hRpt*:

```
setDataDatabase (hRpt, (LPSTR)"accounting");
```

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 RSW.INI with the value specified by *lpszDir*, for the report specified by *hReport*. ARPEGGIO 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");
```

setDataSource

BOOL FAR PASCAL setDataSource (int *hReport*, **LPSTR** *lpszDataSource*);

hReport	Report handle.
lpszDataSource	Data-source buffer.

Return Value

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

Description

Use **setDataSource** to change the ODBC “data source” parameter for the report specified by *hReport*.

Related Functions

getDataSource, **chooseDataSource**

Example

To set the data source for the report whose handle is *hRpt* to "SQLServer QE - ARPEGGIO basic config":

```
setDataSource (hRpt, (LPSTR)"SQLServer QE - ARPEGGIO basic config");
```

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 generated by ARPEGGIO 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, ARPEGGIO will display a status window while it is generating the report; otherwise it will display an icon while it is running. By default, ARPEGGIO will not display status. 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. Note that pressing Cancel will *not* interrupt execution of the Viewer during processing of a SELECT statement by a server.

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

Related Functions

getExportDest, **setOutputFile**

Example

To indicate that the cross-tab 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*. ARPEGGIO 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 ARPEGGIO 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 ARPEGGIO evaluates AND before OR.

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. ARPEGGIO 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 report execution. If no filter was saved with the report, the Insert Condition dialog displays, as shown in Figure 4.1

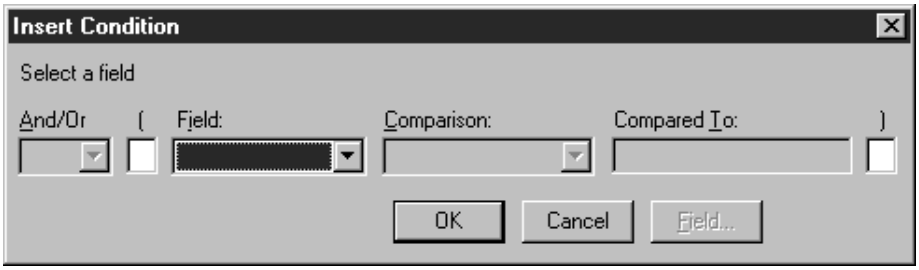

Figure 4.1 Insert Condition Dialog Box

If a filter was saved with the report, the Filter dialog box displays, as shown in Figure 4.2

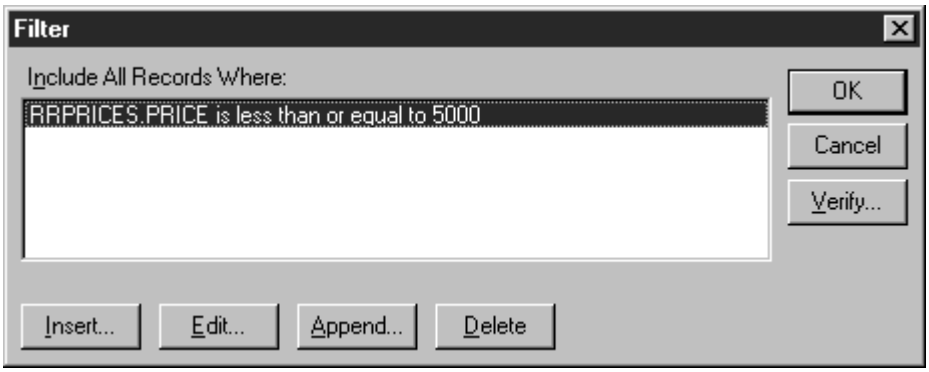

Figure 4.2 Filter Dialog Box

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

Note that the filter-usage parameter has no impact on the value specified by setWhere. If you have used setWhere to specify a WHERE clause, it will always be evaluated by your SQL software directly; any filter will be applied to the result.

Related Functions

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

Example

To allow the user to specify a filter at report execution 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);
}
```

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 RSW.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");
```

setJoinInfo

BOOL FAR PASCAL setJoinInfo (int *hReport*, **LPSTR** *lpszTable*, **LPSTR** *lpszAlias*, **int** *aliasNum*);

hReport	Report handle.
lpszTable	Related table name.
lpszAlias	Alias.
aliasNum	Join-override number.

Return Value

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

Description

Use **setJoinInfo** to replace a related table in the report specified by *hReport*. Use *lpszTable* to specify the new related table name and *lpszAlias* to specify the alias of the related table being replaced. Use an *aliasNum* between 1 and 99 to identify which alias parameter is to be used for the replacement.

Related Functions

getFirstJoinInfo, **getNextJoinInfo**, **setMasterTable**, **getMasterTable**

Example

Suppose the report specified by *hReport* includes a related table fy94, whose alias is fy. If you wish to use **setJoinInfo** to replace fy94 with fy95, you might call **setJoinInfo** as follows:

```
setJoinInfo (hReport,    // report handle
    (LPSTR)"fy95",       // table name
    (LPSTR)"fy",         // alias
    1);                  // number
```

which uses an *aliasNum* of 1 to replace fy94 data with fy95 data. Note that the *lpszAlias* value must match the alias of the related table as saved with the report. The *aliasNum* argument has no significance except to give an ID to the join override specification. If you later realized that you should have used fy93 data you would again call **setJoinInfo** using the same *aliasNum* value of 1. To override the parameters of a different join without losing the fy override, use an *aliasNum* of 2 for the second override.

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* to 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 library directory specified in RSW.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 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 library directory specified in RSW.INI with the value specified by *lpszDir*, for the report specified by *hReport*. Viewer may use the default library directory in trying to locate the library 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");
```

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 columns in the master table specified by *lpszTable* must match in name, number, and type those in the original master table.

Related Functions

getMasterTableName, chooseTable

Example

To specify the use of the table, EMPLOY, for the report whose handle is *hRpt*:

```
setMasterTable (hRpt, (LPSTR)"employ");
```

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 RSW.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*, **char** *cDest*);

hReport	Report handle.
cDest	Output destination.

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 these 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 *cDest* 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 report execution. When the value of *cDest* is a question mark, the user will see the dialog box shown in Figure 4.3. If a title has been specified via **setWinTitle**, the title bar will contain that title; otherwise, the title bar will contain the report name.

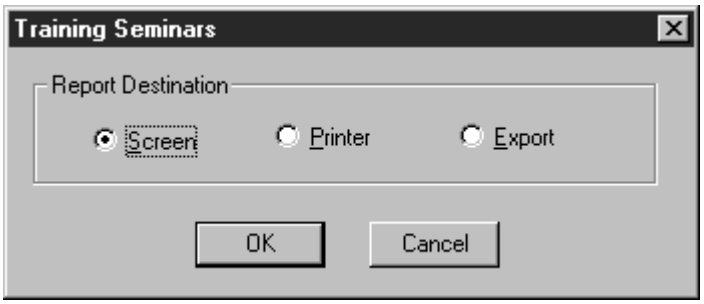

Figure 4.3 Print Destination Dialog Box

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, simply don't 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");
```

setPassword

BOOL FAR PASCAL setPassword (int *hReport*, **LPSTR** *lpszPassword*);

hReport	Report handle.
lpszPassword	Password buffer.

Return Value

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

Description

Use the **setPassword** function to specify the password to be used in connecting to the data source associated with the report specified by *hReport*, either as saved in the report library or as overridden by a call to **setDataSource**. A password is only required for certain data sources. The string pointed to by *lpszPassword* will be used as the password. If **setPassword** is not called to specify a password and the data source requires one, the user will be prompted for a user name and password when the Viewer retrieves the report.

Related Functions

setUserName

Example

To specify "swordfish" as the password and "qawagstaff" as the username for the report whose handle is *hRpt*:

```
setPassword (hRpt, (LPSTR)"swordfish");
setUserName (hRpt, (LPSTR)"qawagstaff");
```

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 Viewer 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 two 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. When the *lpszPrinter* value is a question mark, the Print dialog will display, as shown in Figure 4.4.
- 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 Viewer report(s)

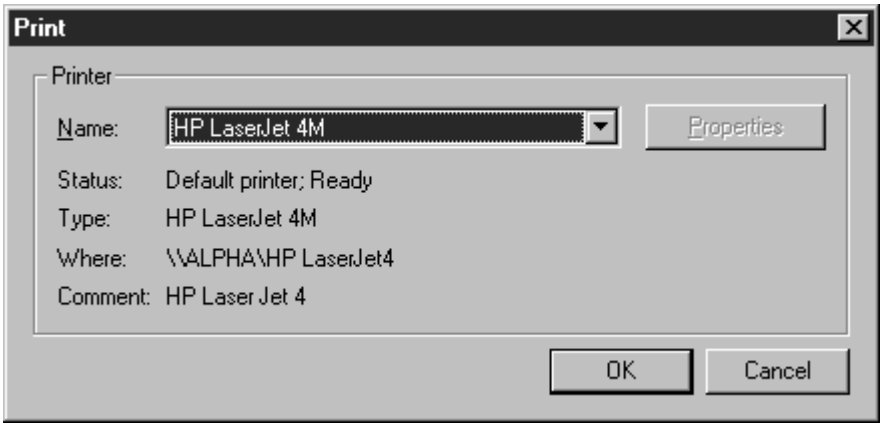

Figure 4.4 Print Dialog Box

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 4.4. 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)"?");
```

setReplace

BOOL FAR PASCAL setReplace (int *hReport*, **LPSTR** *lpszReplace*);

hReport	Report handle.
lpszReplace	Replacement string.

Return Value

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

Description

Use **setReplace** to replace a portion of the SELECT, EXEC, or DEFINE REPORTVIEW statement associated with the User-SQL report specified by *hReport* with the text pointed to by *lpszReplace*.

When you enter a SELECT, EXEC, or DEFINE REPORTVIEW statement in interactive ARPEGGIO, you must enclose in double angle brackets (<< >>) any portion that you may want to replace at report execution. Using **setReplace**, you can provide substitute values for the delimited portions, leave them intact, or specify that you want them to be removed. You can delimit any text in the statement except the initial commands SELECT, EXEC, and DEFINE REPORTVIEW, which cannot be replaced. The initial SELECT, EXEC, or DEFINE REPORTVIEW must be followed by a space. Note also that nesting parameters is not allowed — do not insert delimiters within delimiters.

The syntax of *lpszReplace* is a comma-separated list of parameters enclosed in double angle brackets:

<<param1>>,<<param2>>,<<param3>>,...<<paramN>>

The number of parameters in the *lpszReplace* value ***must match exactly*** the number of delimited portions of the SELECT, EXEC, or DEFINE REPORTVIEW statement saved with the report. Everything between delimiters will be substituted exactly as entered in place of the corresponding delimited text in the original statement. Space outside delimiters is ignored.

Beware of replacing the columns returned by the SELECT statement saved with the report; columns specified by the SELECT as modified by **setReplace** should correspond in name, number, and data type to those specified by the saved SELECT. Otherwise, Viewer will report an error when it cannot find columns referred to in the saved report. If you want to select differently named columns at report execution, assign column names either directly in the SELECT (if this is supported by your database platform) or using the DEFINE

REPORTVIEW syntax. The column names should be the same in the original statement and the *lpszReplace* string.

Note that **setReplace** does not apply to Auto-SQL reports (reports created by selecting master and related tables). To insert a WHERE clause in the SQL statement for an Auto-SQL report, use the **setWhere** function.

Related Functions

getFirstReplace, **getNextReplace**, **setWhere**

Example

If the User-SQL report whose handle is *hRpt* contains the following SELECT statement to select rows for a report:

```
SELECT *
FROM customers
WHERE state='MA'
ORDER BY last_name
```

You can delimit any parts of the statement except the initial word SELECT. For example, you might delimit the FROM, WHERE, and ORDER BY clauses, as shown in this example:

```
SELECT *
<<FROM customers>>
<<WHERE state='MA'>>
<<ORDER BY cust_name>>
```

To provide substitutions for the three delimited sections of the SELECT statement, you might call **setReplace** as follows:

```
setReplace (hRpt, (LPSTR)"<<FROM customers,sales>>,
   <<WHERE customers.cust_no=sales.cust_no AND state='CA'>>,
   <<ORDER BY sale_date>>");
```

To leave any delimited portion intact, use a comma as a place holder. To replace the WHERE clause and leave the FROM and ORDER BY clauses intact, you might call **setReplace** as follows:

```
setReplace (hRpt, (LPSTR)",<<WHERE state='CA'>>,");
```

When you do not want a delimited portion of the statement to be applied, use empty delimiters (<< >>) to specify a null replacement value. For example, this call to **setReplace** specifies that the FROM clause of the original SELECT should be left intact, and the WHERE and ORDER BY clauses should be ignored:

```
setReplace (hRpt, (LPSTR)",<<>>,<<>>");
```

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');
```

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];

   getFirstGroupField (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 Viewer will generate a status file. If *bWait* is non-zero, no Viewer status file is generated and status is returned to the calling application via **execRuntime**. If *bStatus* is non-zero and Viewer is generating a status file, 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 Viewer 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 RSWRUN.OUT 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.OUT for the report specified by *hRpt*:

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

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);
```

setUserName

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

hReport	Report handle.
lpszName	User-name buffer.

Return Value

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

Description

Use the **setUserName** function to specify the user name to be used in connecting to the data source associated with the report specified by *hReport*, either as saved in the report library or as overridden by a call to **setDataSource**. A user name is required only for certain data sources. The string pointed to by *lpszName* will be used as the user name. If **setUserName** is not called to specify a user name and the data source requires one, the user will be prompted for a user name and password when the Viewer retrieves the report.

Related Functions

setPassword

Example

To specify "swordfish" as the password and "qawagstaff" as the username for the report whose handle is *hRpt*:

```
setPassword (hRpt, (LPSTR)"swordfish");
setUserName (hRpt, (LPSTR)"qawagstaff");
```

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 Viewer. The value of a user parameter is obtained within a report via the ARPEGGIO function RIPARAM. 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 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 ARPEGGIO Report Designer 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:". As a result, the user will see the dialog box shown in Figure 4.5.

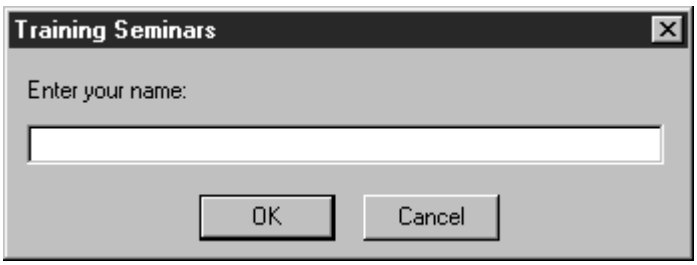

Figure 4.5 Viewer Dialog Box with Prompt

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 4.5 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**

setWhere

BOOL FAR PASCAL setWhere (int *hReport*, **LPSTR** *lpszWhere*);

hReport	Report handle.
lpszWhere	Where-clause value.

Return Value

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

Description

Use **setWhere** to specify a WHERE clause, pointed to by *lpszWhere*, for insertion in the SQL statement of the Auto-SQL report specified by *hReport*. **setWhere** has no effect on a User-SQL report. Use **setReplace** to modify the SELECT statement of a User-SQL report.

If you or your users are proficient in SQL, you may want to use this function instead of **setFilter** and **setFilterUsage** to select records. Since the WHERE clause is evaluated directly by the SQL software, using **setWhere** can improve performance and enable you to make use of any WHERE clause supported by your SQL software.

The WHERE clause specified with this function always affects the report, regardless of whether a filter was saved with the report. If you have also used **setFilter** and **setFilterUsage** to select records, the effect of *lpszWhere* is as follows:

- If **setFilterUsage** specifies **S** for "Saved", both the filter saved with the report *and* the *lpszWhere* clause are used to select records.
- If **setFilterUsage** specifies **O** for "Override", both the filter expression specified via **setFilter** *and* the *lpszWhere* clause are used to select records.
- If **setFilterUsage** specifies **E** for "Entire," *only* the *lpszWhere* clause is used to select records.
- If **setFilterUsage** specifies a question mark (**?**) to allow the user to enter a filter interactively, both the user's filter expression *and* the *lpszWhere* clause are used to select records.

Related Functions

setFilter, **setFilterUsage**, **setReplace**, **getFirstReplace**, **getNextReplace**

Example

To specify that the report whose handle is *hRpt* should be run with its saved filter and with the additional clause, "where STATE is MA":

```
setFilterUsage (hRpt, 'S');
setWhere (hRpt, (LPSTR)"STATE='MA'");
```

Note that it is not necessary to know what the saved filter is. This sample will further restrict the set of records to those from the state of Massachusetts.

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 **setStatusEveryPage** is called with a non-zero *bStatus* value.);
- Below the Viewer icon (if **setStatusEveryPage** 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.

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

Related Functions

getWinTitle, setStatusEveryPage, setPrinter

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);
```

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 5
Using the Custom Control (OCX)

Introduction

As noted in Chapter 1, the Viewer OCX (or custom control) provides one of three methods for running reports using the Report Viewer. The other methods are explained in Chapter 2, "Using the Report Viewer," and Chapter 4, "Accessing the Viewer DLL."

Like the report control in the Professional Edition of VB, the Viewer OCX allows you to incorporate database reporting into your applications. However, it provides the following additional advantages:

- Access to the more powerful reporting capabilities of ARPEGGIO Report Designer;
- More control over how your report is printed by means of over 50 properties that can be set at design time or run time;
- Far more 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 RSW.INI for Default Information
- Using the OCX
- Custom Control Properties

Installation

The OCX is installed if you select a Setup Type of "Typical" or if you select "Custom" and specify Report Viewer as one of the options.

To add the OCX to an existing VB project, select Tools ⇒ Custom Controls to add the file RSW32.OCX, which Setup installs into your Windows System directory. For information about other files you will

need to distribute when you use the ARPEGGIO OCX in your applications, see Chapter 8, "Distributing Reports."

Setup also copies two sample applications into appropriate subdirectories of the ARPEGGIO SAMPLE directory:

- OCXTEST, which is installed in SAMPLE\MFCOCX, demonstrates a common way to use an OCX control in a MFC C++ program.
- RSWVB32, which is installed in SAMPLE\VB, demonstrates the use of the OCX in Visual Basic code.

Each sample has an accompanying README file that explains it in more detail.

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 custom 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. RSReport1.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. RSReport1.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 by default a text file called RSWRUN.OUT in the current working directory. You can change the path and name of the status file via the StatusFileName property. (See **Understanding the Viewer Status File** in Chapter 2 for a description of the contents of the status file.)

Using RSW.INI for Default Information

Report Designer stores log-on information in the file RSW.INI, which is created in the Windows directory when you install ARPEGGIO. When you start Report Designer, the log-on information (except the password) is saved in RSW.INI. The default log-on information for each database platform is replaced whenever you connect to that platform. The Viewer will look for an RSW.INI file in the Windows directory and attempt to use the defaults if the custom control does not supply the log-on information. Any information you supply using custom control properties will always override the RSW.INI setting (see Figure 2.2 in Chapter 2 for a list of RSW.INI settings that the Viewer will use).

If you distribute reports to other users, you can customize RSW.INI for each user and distribute it with the other Viewer files. However, the custom control properties provide a more reliable way to supply accurate and up-to-date log-on information.

Using the OCX

You use the ARPEGGIO OCX just like any other OCX. Simply click on the ARPEGGIO 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 an OCX 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 OCX will be visible on your VB form 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 4). As you will see, the OCX properties correspond closely to the control parameters used by the Viewer. (See Appendix A, "Viewer Equivalencies.")

To print or display a report in your VB 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., RSReport1.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 Properties list to set or change values, the data type of the property determines how you change its value:

- You change an integer property by simply typing a value into the settings box.
- You change an enumerated property by typing a number into the settings box or by selecting 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 5.1) consists of 12 property pages that you can use to control any of the more than 50 custom control properties.

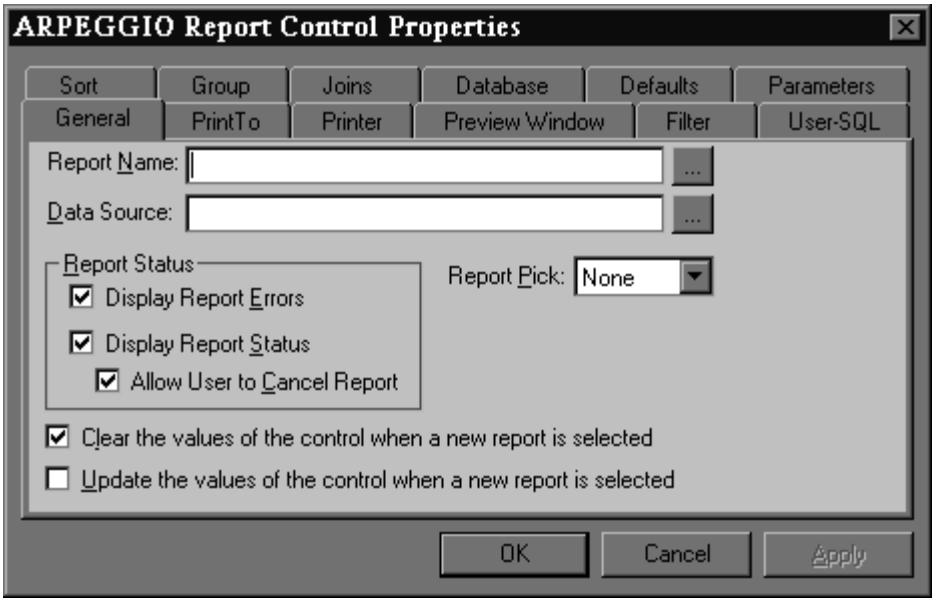

Figure 5.1 Report Control Properties Dialog

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.

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 ellipsis button to display the Open dialog for selection of a report file.
Data Source	Sets the **DataSource** property to specify or override the data source for the report(s).
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 are reset to the saved values 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 report execution.

Print To Property Page

Settings on the Print To property page control or override the print and/or export destination of the 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, 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 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 report execution 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 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.

Filter Property Page

The Filter property page specifies or overrides the filter to be used to filter the report data.

Setting	Purpose
Saved Filter	Sets the **Filter** property to specify using the saved filter, ignoring the saved filter, overriding the saved filter, or prompting the user.
Where	Sets the **Where** property to modify the WHERE clause of the SQL statement in an Auto-SQL report.

User-SQL Property Page

The User-SQL property page specifies replacement strings for those portions of the report's SELECT statement that have been marked as replaceable.

Setting	Purpose
Replace	Lists replaceable strings from the report's SELECT statement.
With	Sets the **Replace** property to modify the SELECT statement in a User-SQL report.

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. By default, sorting is in ascending order; to change to descending order, click the Ascending box to remove the checkmark.

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.

Joins Property Page

The Relations property page specifies or overrides the related tables for the report.

Setting	Purpose
Alias	Lists aliases of the related tables.
Table	Sets the **RelatedTables** property to specify or override table joins.

Database Property Page

The Database property page provides settings for master table selection and for the name and location of the text memo file to be used by the report.

Setting	Purpose
Master Table	Sets the **MasterTable** property to override the master table saved with the report.
Memo File	Sets the **MemoFileName** property to specify the name and location of the text memo file to be used by the report.

Defaults Property Page

The Defaults property page specifies or overrides defaults for data, image, and report directories.

Setting	Purpose
Default Report Directory	Sets the **ReportDirectory** property to specify default location for report or library files.
Default Image Directory	Sets the **ImageDirectory** property to specify default location for image files used by reports.
Default Data Directory	Sets the **DataDirectory** property to specify default locations for Xbase tables, Paradox tables, and text memo files.

Parameters Property Page

The Parameters property page sets the **ParametersString** property to specify user parameter values to be used with the 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 report execution.

Custom Control Properties

The following properties are available in the ARPEGGIO Custom Control:

(About)

Description

Double-click About to display the version of the ARPEGGIO 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

```
RSReport1.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.

If it is 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 it is 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 does complete, its status will be written into the text file RSWRUN.OUT in the current working directory(or to the path specified with the StatusFileName property).

In most cases, you will find it more convenient to set this property to 1.

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

```
RSReport1.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

Database

Description

For database platforms that support multiple databases, you can use this property to specify (or override) the database for the report tables.

Usage

[form.]*ControlName*.Database[= DatabaseName$]

Example

```
RSReport1.Database = "sales"
```

« Use tables in the database named “sales.” »

Comments

SQL Server is one platform that supports multiple databases.

Data Type

String

Availability

Run time

DataDirectory

Description

Specifies the default directory where the Viewer will look for Xbase and Paradox tables and text memo files to be used when the report is printed.

Usage

[form.]*ControlName*.DataDirectory[= DirectoryName$]

Example

```
RSReport1.DataDirectory = "c:\mis\data"
```

« Looks for data files in a directory called “c:\mis\data.” »

Comments

If the Xbase and Paradox tables 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

DataSource

Description

Specifies the ODBC data source to be used to access the report data.

Usage

[form.]*ControlName*.DataSource[= DataSourceName$]

Example

```
RSReport1.DataSource = "dBASE MS - ARPEGGIO Sample"
```

« Use tables in the data source named “dBASE MS - ARPEGGIO Sample.” »

Comments

This parameter is optional. The data source should contain the same table(s) as the data source originally used in the report.

If you leave this property blank, the Viewer uses the data source saved with the report.

At design time, you can change this property in two ways:

- Double-click this property to display the General property page. Click the Data Source ellipsis button to display the Data Source Connections dialog (see Figure 5.2); then highlight the data source and select OK.
- Simply enter the data source name into the settings box.

Data Type

String

Availability

Design time; Run time

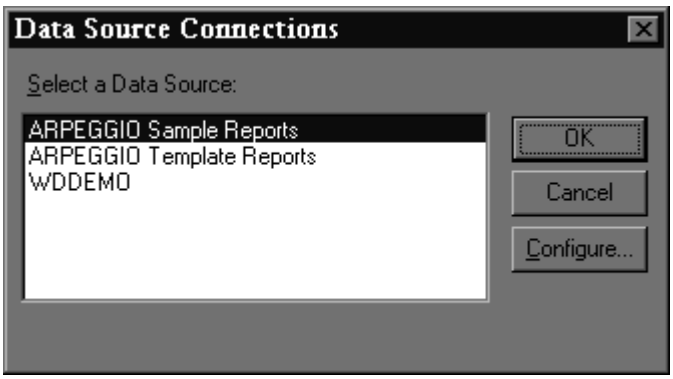

Figure 5.2 Data Source Connections Dialog Box

Destination

Description

Specifies the destination to which your report is to be printed or exported (Preview Window, Printer, Text File, DBF File, WKS File, or RTF File).

Usage

[form.]*ControlName*.Destination[= Destination%]

Example

```
RSReport1.Destination = 1
```

« Sends the report to a preview window. »

Comments

This property is optional. Set the property to 0 (the default) to print to the printer saved with the report (or to the printer specified in the Printer property). It 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

If you specify 3 (Text File), 4 (DBF File) 5 (WKS File), 7 (RTF File), 8 (Text Data file), or 9 (Word Merge file), you can also set the PrintFileName property to provide the name of the destination file to override the saved destination file name.

If you specify 1 (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 tool 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.

To print the report to a text file that includes embedded printer codes, set Destination to 0 (saved) and specify an output file name with PrintFileName.

Setting this property to 6 (Prompt user) allows the user to select the print destination at run time. The user will see the dialog box shown in Figure 5.3. 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.

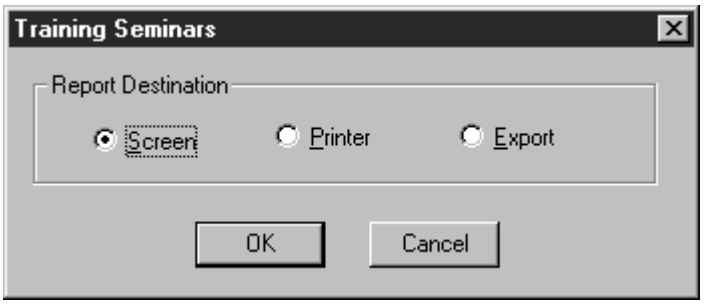

Figure 5.3 Print Destination Dialog Box

The user can select Screen to preview the report; Printer to print it; or Export to display a dialog for selection of an export type (Excel Chart, Excel PivotTable, RTF, Text, Text Data, Word Merge, Xbase, or

Worksheet). 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 or not errors are to be displayed when a report is printed.

Usage

[form.]*ControlName*.DisplayError [= {True|False}]

Example

```
RSReport1.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 terminate processing.

If DisplayError is False, Viewer error messages are not displayed, but are returned in the LastErrorString property or the status file.

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

```
RSReport1.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 program 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. Note that pressing Cancel will *not* interrupt execution of the Viewer during processing of a SELECT statement by a server.

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

```
RSReport1.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 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

ExportDestination

Description

Specifies the destination (display, file, or printer) when exporting to an Excel PivotTable or Chart.

Usage

[form.]*ControlName*.ExportDestination[= Destination%]

Example

```
RSReport1.ExportDestination = 2
```

« Exports the report to a file. »

Comments

Set this property to one of the following values if you are exporting to Excel PivotTable or Excel 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 filter to select records to be used when printing the report.

Usage

[form.]*ControlName*.Filter[=Filter$]

Example

```
RSReport1.Filter = "Year > 1994"
```

Comments

The optional Filter property specifies a logical expression that will override the saved filter, if any, when the value in Include 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 filter 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 filter expression **LIKE("*Dallas*",NOTE)** 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 greater than or equal to 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 filter expressions can be entered by using parentheses. For example, the following filter 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 Include *must* be 2 in order for the Filter override to take effect. If you omit Include, the Filter value will be ignored and the report will be run using the saved filter (if any).

When setting this property at run time, make sure that you enclose your filter expression in double quotes. If your filter 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 filter expression, such as:

"LNAME = ‘Jones’"

At design time, you can change this property array in two ways:

- Double-click this property to display the Filter property page (see Figure 5.4).

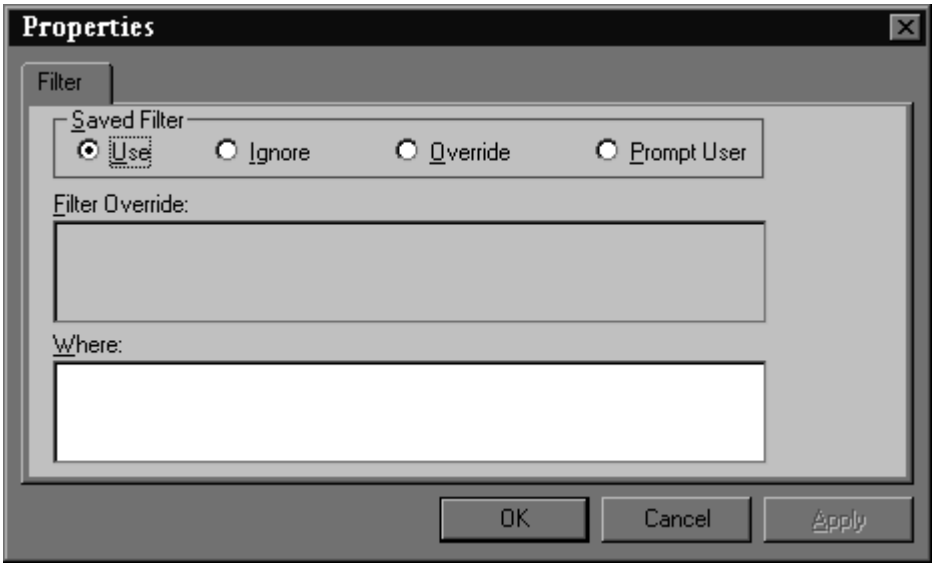

Figure 5.4 Filter Property Page

- Simply enter the filter 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

```
RSReport1.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

```
RSReport1.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

ImageDirectory

Description

Specifies the default directory where the Viewer may look for image files used in the report.

Usage

[form.]*ControlName*.ImageDirectory[= DirectoryName$]

Example

```
RSReport1.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 will override any default image directory specified in the RSW.INI file.

Data Type

String

Availability

Run time

Include

Description

Specifies which, if any, filter to use when the report is run.

Usage

[form.]*ControlName*.Include [= QueryOption%]

Example

```
RSReport1.Include = 2
```

« Ignores the filter in the report and uses the filter expression in the Filter property in place of it. »

Comments

The optional Include property allows you to control whether a filter is applied to the report. Include can have one of four values:

0 – Saved. Run the report using the filter 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 filter saved in the report or contained in the Filter property.

2 – Override. Override the saved filter 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 filter expression or edit the filter saved with the report. If no filter was saved with the report, the Insert Condition dialog will display, as shown in Figure 5.5.

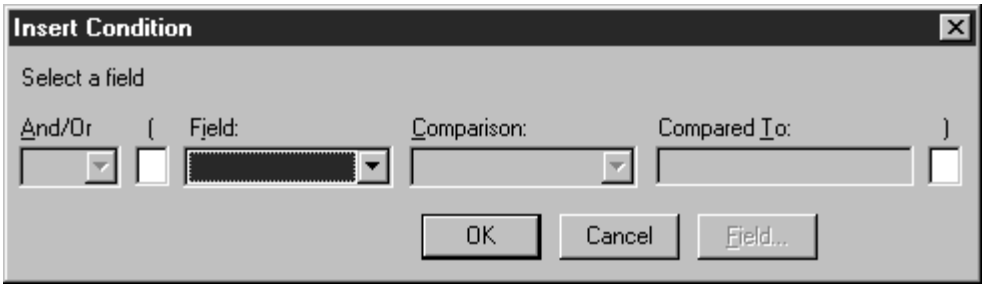

Figure 5.5 Insert Condition Dialog Box

If a filter was saved with the report, the Filter dialog box will display, as shown in Figure 5.6.

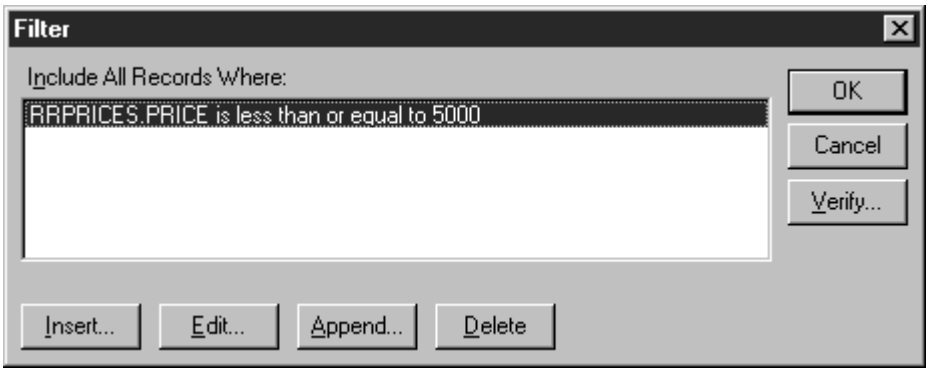

Figure 5.6 Filter Dialog Box

When you set Include to 3, the value of the Filter property is always ignored.

Note that Include has no impact on the Where property. If the Where property is specified, it will always be evaluated by your SQL software directly; any filter will be applied to the result.

Data Type

Integer (Enumerated)

Availability

Design time; Run time

LastErrorCode

Description

Returns the error code for the last Viewer error. It will be one of the following four values:

0 = No error;

1 = User canceled;

2 = Error in Viewer parameters;

3 = Error in report.

Usage

[form.]*ControlName*.LastErrorCode

Example

```
'If error occurs, display error message
RSReport1.Action = 1
if RSReport1.LastErrorCode <> 0 then
      MsgBox  RSReport1.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 report page.

Usage

[form.]*ControlName*.LastErrorPage

Example

```
'If error occurs, display error message
RSReport1.Action = 1
if RSReport1.LastErrorCode <> 0 then
   pagestr$ = "; last page printed was "
   + str(RSReport1.LastErrorPage)
   MsgBox  RSReport1.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 Viewer error.

Usage

[form.]*ControlName*.LastErrorString

Example

```
'If error occurs, display error message
RSReport1.Action = 1
if RSReport1.LastErrorCode <> 0 then
      MsgBox  RSReport1.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 OCX controls with the current report settings.

Usage

[form.]*ControlName*.LoadProperties( )

Example

```
RSReport1.LoadProperties()
```

«Updates the OCX controls with the settings from the current report. »

Comments

This method is used to load all OCX properties with the values from the current report; use it to display or explicitly see all report properties.

Availability

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

```
RSReport1.MasterTable = "school.dbo.students"
```

« Uses the table “school.dbo.students” as the master table. »

Comments

This property is optional. The master table you specify should have the same columns as the master table used in the report. If you leave this property blank, 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 5.7).

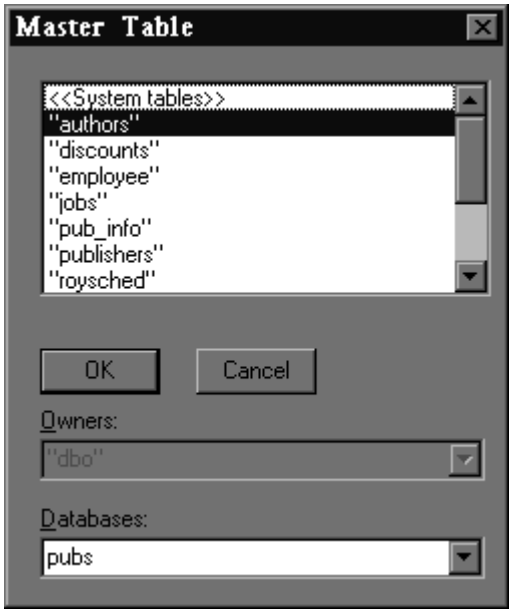

Figure 5.7 Master Table Dialog (SQL Server)

- Simply enter the table name into 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

```
RSReport1.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 RSW.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

```
RSReport1.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 pressing Cancel will *not* interrupt execution of the Viewer during processing of a SELECT statement by a server.

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 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

```
RSReport1.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 Chapter 3, “Parameter Passing,” for information on how to use 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

```
RSReport1.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

Password

Description

Enters the password needed to use database tables on a password-protected SQL database.

Usage

[form.]*ControlName*.Password[= Password$]

Example

```
RSReport1.Password = "brokencrystal"
```

« Enters the password “brokencrystal.” »

Comments

If a valid password and user name are not provided, the user will be prompted to enter them when the report is run.

Data Type

String

Availability

Run time

Port

Description

Specifies the name of the printer port to which the report is to be printed.

Usage

[form.]*ControlName*.Port [= PortName$]

Example

```
RSReport1.Port = "LPT1:"
```

« Prints the report to the printer port named “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 5.8. 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

```
RSReport1.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 Report Designer Print dialog (accessed by selecting File ⇒ Print in Report Designer). 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 report execution. When the Printer property contains a question mark, the Print dialog will display, as shown in Figure 5.8.
- 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)

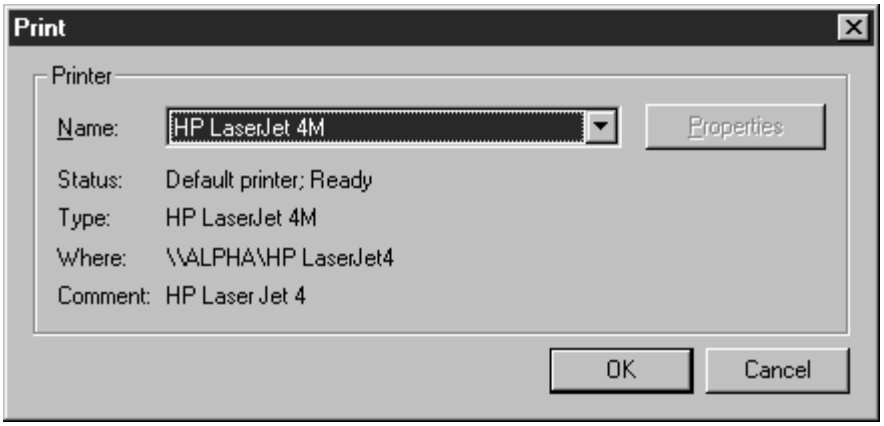

Figure 5.8 Print Dialog Box

Initially, the printer saved with the report is highlighted. The user can select another printer 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 (Use Saved Printer, Prompt User, or Override Saved Printer).
- 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

```
RSReport1.PrintFileName = "c:\output\q3sales.txt"
```

« Prints the report to a file named “q3sales.txt” in the c:\output directory. »

Comments

Use this property if you have set the Destination property to 3 (Text File), 4 (DBF File) 5 (WKS File), 7 (RTF File), 8 (Text Data file), or 9 (Word Merge file) 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 filename into the settings box.

Data Type

String

Availability

Design time; Run time

RelatedTables

Description

Specifies table joins to override those saved with the report.

Usage

[form.]*ControlName*.RelatedTables(ArrayIndex)
[= Alias$=TableName$]

Example

```
RSReport1.RelatedTables(0) = "FIRST=dbo.students"
RSReport1.RelatedTables(1) = "SECOND=dbo.grades"
```

« Changes the first and second table joins in the report. »

Comments

These properties are optional. If you do not specify any table join overrides, the Viewer uses the table joins saved with the report. It searches for the related tables using the search rules explained in Chapter 8.

When setting this property at run time, use a separate line of code for each change. Up to nine (9) different table joins 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

```
RSReport1.RelatedTablesString =
"FIRST=c:\q2\first.dbf; SECOND=c:\q2\second.dbf"
```

« Changes the first and second table joins 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 8.

At design time, you can change this property array in two ways:

- Double-click this property to display the Joins property page. Then use the ellipsis buttons to select related tables. 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

Replace

Description

Specifies replacement strings for a User-SQL report.

Usage

[form.]*ControlName*.Replace [= <<String1$>>,<<String2$>>,etc.]

Example

```
RSReport1.Replace = "<<from dbo.students>>,,
<<order by ST_PROV,CITY>>,<<>>"
```

« Changes the first and third replacement string in the User-SQL statement, leaves the second one unchanged, and causes the fourth one to be ignored. »

Comments

The optional Replace property allows you to supply a substitute value to override all or part of the SELECT, EXEC, or DEFINE REPORTVIEW statement used to select rows for a User-SQL report.

When you enter a SELECT, EXEC, or DEFINE REPORTVIEW statement in Report Designer, you must enclose in double angle brackets (<< >>) any portion that you may want to replace at run time. Using Replace, you can provide substitute values for the delimited portions, leave them intact, or specify that you want them to be ignored at run time. You can delimit any text in the statement except the initial commands SELECT, EXEC, and DEFINE REPORTVIEW, which cannot be substituted. The initial SELECT, EXEC, or DEFINE REPORTVIEW must be followed by a space. Note also that nesting parameters is not allowed — do not insert delimiters within delimiters.

The syntax of Replace is a comma-separated list of parameters enclosed in double angle brackets:

```
<<param1>>,<<param2>>,<<param3>>,...<<paramN>>
```

The number of parameters in the Replace value *must match exactly* the number of delimited portions of the SELECT, EXEC, or DEFINE REPORTVIEW statement saved with the report. Everything between delimiters will be substituted exactly as entered in place of the corresponding delimited text in the original statement. Space outside delimiters is ignored.

For example, you may be using this SELECT statement to select rows for a report:

```
SELECT *
FROM customers
WHERE state='MA'
ORDER BY last_name
```

You can delimit any parts of the statement except the initial word SELECT. For example, you might delimit the FROM, WHERE, and ORDER BY clauses, as shown in this example:

```
SELECT *
<<FROM customers>>
<<WHERE state='MA'>>
<<ORDER BY cust_name>>
```

To provide substitutions for the three delimited sections of the SELECT statement, you might supply the following value in the Replace property:

```
<<FROM customers,sales>>,
<<WHERE customers.cust_no=sales.cust_no
  AND state='CA'>>,
<<ORDER BY sale_date>>
```

To leave any delimited portion intact, use a comma as a place holder. To replace the WHERE clause and leave the FROM and ORDER BY clauses intact, you might use this Replace value:

```
,<<WHERE state='CA'>>,
```

When you do not want a delimited portion of the statement to be applied, use empty delimiters (<< >>) to specify a null replacement value. For example, this Replace value specifies that the FROM clause of the original SELECT should be left intact, and the WHERE and ORDER BY clauses should be ignored:

```
,<<>>,<<>>
```

In general, application of the Replace property must yield a SELECT statement that would itself be valid as the basis of an ARPEGGIO User-SQL report. For example, all columns in the result of the modified SELECT must be uniquely named. In addition, any columns returned by the original User-SQL SELECT that are used in the report must also be returned by the modified SELECT with the same names and types.

Note that Replace values are not applied to Auto-SQL reports (reports created by selecting master and related tables). To insert a WHERE clause in the SQL statement for an Auto-SQL report, use the Where property.

At design time, you can change this property in two ways:

- Double-click this property to see the User-SQL property page (see Figure 5.9), which contains a list of replacement strings in the report.

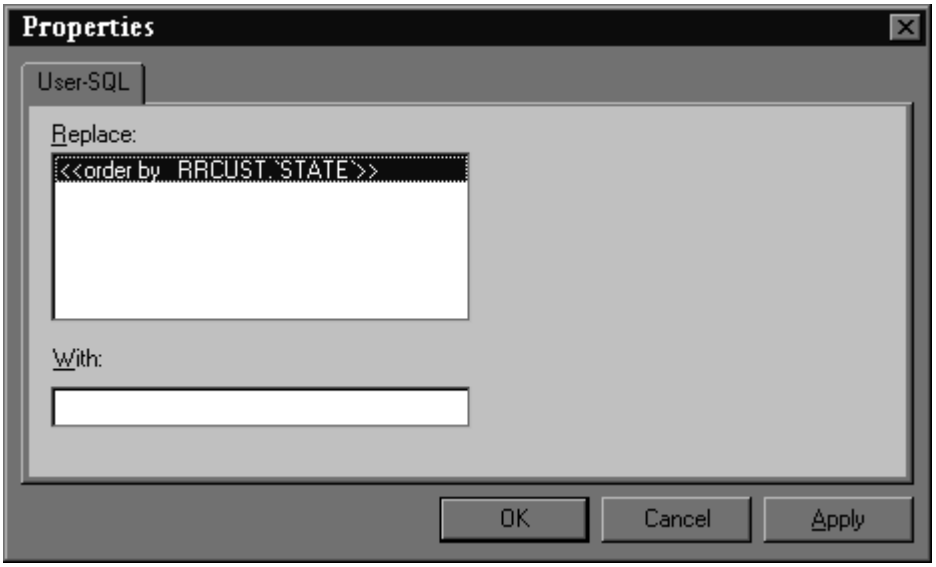

Figure 5.9 Replace User-SQL Property Page

- Enter the names of each replacement string separated by commas. If you want to override some replacement strings, but not all of them, you must use a comma as a place-holder.

Data Type

String

Availability

Design time; Run time

ReportDirectory

Description

Specifies a default directory where the Viewer may look for the report library specified in ReportName or ReportLibrary.

Usage

[form.]*ControlName*.ReportDirectory[= DirectoryName$]

Example

```
RSReport1.ReportDirectory = "c:\mis\reports"
```

« Looks for the report in a directory called “c:\mis\reports.” »

Comments

If the report file 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 library directory specified in the RSW.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

```
RSReport1.ReportLibrary = "c:\rrw\rrsample\rrsample.rp6"
```

« Selects the report library named “rrsample.rp6” in the c:\rrw\rrsample directory. »

Comments

For a report being retrieved from a report library, 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 RSW.INI file. If RSW.INI is not present and no default directory is specified, the Viewer searches for the library in the current directory.

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 to which you have access.
- 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

```
RSReport1.ReportName = "Order Invoice "
```

« 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 see the General Property page. Then click the ellipsis button next to Report Name to display the Open Report dialog (see Figure 5.10), which contains a list of report files in the location specified in the ReportDirectory property.

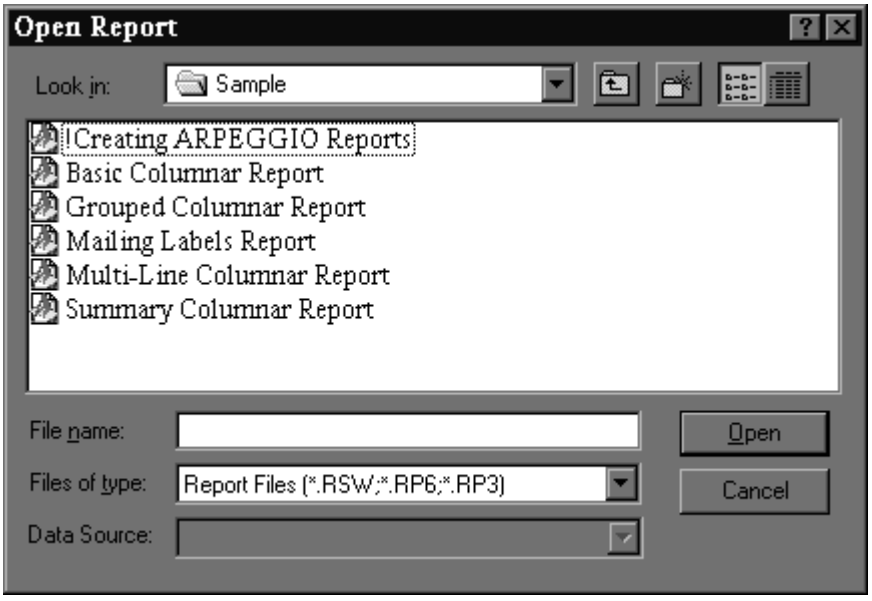

Figure 5.10 Open Report Dialog Box

- 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

```
RSReport1.ReportPick = 1
```

« Displays a list of reports in the location specified by ReportDirectory 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, you do not need to set the ReportName property; if you include both ReportPick and ReportName values, Viewer ignores the ReportName.

Set this property of 2 to have the Viewer prompt the user to select a succession of reports. When the value is 2, Viewer will prompt the user to select a report from those in the location specified by ReportDirectory. 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 presses Esc.

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].*ControlName*.ResetControl( )

Example

```
RSReport1.ResetControl( )
```

Comment

Use this method at report execution 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 OCX should reset its properties when a new report is specified.

Usage

[form].*ControlName*.ResetProperties[= {TRUE|FALSE}]

Example

```
RSReport1. 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

```
RSReport1.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

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

```
RSReport1.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

```
RSReport1.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

```
RSReport1.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 status file.

Usage

[form.]*ControlName*.StatusFileName[=StatusFileName$]

Example

```
RSReport1.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

```
RSReport1.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

```
RSReport1.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

UserName

Description

The user name for logging on to a SQL database.

Usage

[form.]*ControlName*.UserName[= Name$]

Example

```
RSReport1.UserName = "RSmith"
```

« Enters the user name “RSmith.” »

Comments

If a valid password and user name are not provided, the user will be prompted to enter them when the report is run.

Data Type

String

Availability

Run time

Where

Description

Modifies the WHERE clause of the SQL statement in an Auto-SQL report.

Usage

[form.]*ControlName*.Where [= WhereClause$]

Example

```
RSReport1.Where = "(dbo.students.PROGRAM = 'ELEC')"
```

« Changes the WHERE clause in the SQL statement to be "WHERE (dbo.students.PROGRAM = 'ELEC')". »

Comments

The optional Where property enables the Viewer to insert a WHERE clause in the SQL statement for an Auto-SQL report. If you or your users are proficient in SQL, you may want to use this property instead of Filter and Include to select records. Since the WHERE clause is evaluated directly by the SQL software, using the Where property can improve performance and enable you to make use of any WHERE clause supported by your SQL software.

The WHERE clause specified with this property always affects the report, regardless of whether a filter was saved with the report. If you have also used the Filter and Include properties to select records, the effect of the Where property is as follows:

- If Include is "0 – Saved", both the filter saved with the report *and* the clause in Where are used to select records.
- If Include is "2 – Override", both the filter expression in Filter *and* the clause in Where are used to select records.
- If Include is "1 – Entire," *only* the Where clause is used to select records.
- If Include is "3 – Prompt user" to allow the user to enter a filter interactively, both the user's filter expression and the WHERE clause are used to select records.

Note that Where values are not applied to User-SQL reports. To override the selection conditions for a User-SQL report, use the Replace property.

Data Type

String

Availability

Design time; Run time

WindowBorderStyle

Description

Specifies the type of border for the preview window.

Usage

[form.]*ControlName*.WindowBorderStyle[= BorderStyle%]

Example

```
RSReport1.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 hand corner when the report is displayed in a preview window.

Usage

[form.]*ControlName*.WindowControlBox[= {True|False}]

Example

```
RSReport1.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 preview window when the report is displayed in a preview window.

Usage

[form.]*ControlName*.WindowHeight[= Height%]

Example

```
RSReport1.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

```
RSReport1.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 displayed in a preview window.

Usage

[form.]*ControlName*.WindowMaxButton[= {True|False}]

Example

```
RSReport1.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 or not the preview window is to have a minimize button when the report is displayed in a preview window.

Usage

[form.]*ControlName*.WindowMinButton[= {True|False}]

Example

```
RSReport1.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 displayed in a preview window.

Usage

[form.]*ControlName*.WindowTitle[= Title$]

Example

```
RSReport1.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 or Port 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, enclose the title 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

```
RSReport1.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

```
RSReport1.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 = 1).

Data Type

Integer

Availability

Design time; Run time

Chapter 6
ARPEGGIO ReportScript

Introduction

ARPEGGIO provides an open architecture for report generation. Called "ARPEGGIO ReportScript," this new architecture allows developers to create application-specific reporting front ends that use ARPEGGIO Report Designer's sophisticated reporting engine for actual report generation. ARPEGGIO ReportScript allows any developer who can generate text files to communicate effectively with Report Designer.

There are two ways in which you may take advantage of the ReportScript interface to generate a report. The Report Wizards make use of this script mechanism to pass a user-specified report specification from the Report Wizards to the Report Designer executable. In order to provide a custom interface for report creation, developers may configure Report Designer to integrate their own Windows executable (EXE) in place of the ARPEGGIO 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 of this chapter for more information about the script file.

Explanation of ReportScript 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 ReportScript 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 Report Designer is very simple.

You can also use a ReportScript file 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 ARPEGGIO Wizards program with your own, simply add the following settings to the initialization file (RSW.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 ARPEGGIO 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 you select Report Wizards and choose a database table, 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. The arguments are separated by semicolons (;). The first argument is the pathname to an "input file" containing information used by the ARPEGGIO 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 master database table selected within Report Designer. (For desktop database platforms such as Paradox or Xbase, this argument is a complete pathname, including drive specifier. For SQL platforms, this argument is a table name, including database qualifier if the SQL engine allows a database override.) The master table passed on the command line must be used by the custom application as the basis for report generation.

The script file is read by Report Designer 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 performing a print preview of 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 ReportScript 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 duplicate keywords in the same section, which is generally not the case with Windows initialization files.

These are the general rules that apply to the format of script files:

- 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 database field length parameters will be set to the values stored in the database table.
- All field definitions specified will be inserted on the report 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. (Note that you should 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 ARPEGGIO 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 ARPEGGIO.

Script File Sections and Keywords

The report sections and keywords used in this version of Report Designer are indicated below. All sections and keywords are optional, with the exception of the Report section, which is required when a script file is passed to Report Designer on the command line. This section is ignored if specified in a script file generated by a custom application that replaces the ARPEGGIO Report Wizards.

REPORT SECTION
[Report]
MasterTable=tablename

If Report Designer is called with a script file as a command-line argument, the MasterTable argument is used as the table from the specified database to generate the report described in the remainder of the script file.

ACTION SECTION
ReportScript 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.

In the following example of a character field definition to be placed on a Record line, field name is CUSTNAME; *field trim is* True; *field location is* 1.00 inch; *alignment is* left; *style is* Underscored; *and length is* zero *(i.e., 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
1: dd-mmm-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
12: dd/mm/yy
13: dd/mm/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

DATABASE DATE/TIME FIELD
DateTimeField=Name (required),
Trim (T or F),
Location,
Alignment,
Style (BUIS),

Date picture:

0: dd-mmm-yy
1: dd-mmm-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
12: dd/mm/yy
13: dd/mm/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: International

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

ARPEGGIO ReportScript 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 ARPEGGIO Report Wizards is shown on the following page. 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
MasterTable=employee

; 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="*** 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 Wizards. This input file contains information that is useful to users of the ARPEGGIO ReportScript 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 Report Wizards, this file may be ignored by any custom application.

The sections and keywords placed in the input file are indicated below. 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=1 (RSW)
Pitch=12 (Pitch of default font)
PageWidth=7.50 (Default page width minus left and 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
1: Numeric
2: Date
3: Logical
4: Memo
5: Date/Time
6: Time

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. Below 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 7
Interfacing to Application DLLs

Introduction

ARPEGGIO Report Designer includes a special function, CDLL( ), that allows you to call a DLL-based function from a report. You might use CDLL( ) when you want to write a DLL-based function to perform an operation that Report Designer's UDFs don't support, such as a trigonometric function. CDLL( ) also gives you access from reports 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. You can use 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 miles to kilometers.

CDLL( ) expects a boolean return value from the called 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. The input and output strings are passed 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. 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 8
Distributing Reports

Introduction

You can distribute reports to users who have licensed copies of either ARPEGGIO Report Designer or ARPEGGIO Viewer. You and your users must have the same version of the Viewer, preferably the most recent one. Your users must also have the required ODBC driver(s) and the appropriate data source configuration(s) for access to the database files you provide them.

Information about distributing reports is provided in the following sections:

- Required Files for Report Distribution
- Distributing VB Applications
- Retrieving Report Files

Required Files for Report Distribution

Figure 8.1 lists files to be distributed in order to enable users to access reports. You can specify locations for some of these files either with command switches or with the Viewer control fields.

Note that ARPEGGIO saves with each report the relevant data source information and the location of tables, image files, and text memo files. The Viewer will automatically find these files if they are retrievable according to the search rules explained in the **Retrieving Report Files** section. The Viewer will also look for these files in the default directories specified on the Viewer command line or in the RSW.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 RSW.INI available to users, it should be stored in the user's Windows directory. The UDF library file, RSW.UDF, and RSWSQL.INI should be in the same directory as the Viewer executable, RSWRUN.EXE.

Note that if your users will be exporting reports to Excel PivotTable or Excel Chart, they must have Excel 5.0 (or later) installed.

File	*Description*	*Location*
Report File(s)	The report(s) to be run by the Viewer.	Any directory
Data files	The tables, indexes, text memo files, and image files used by your reports (if your users do not already have them). All data files used by your report(s) are required.	Any directory
Control Table or File	The database table or text file that provides the Viewer with information about the report(s) to be run.	Any directory
RSW.UDF	User-defined function file. Required if your reports use any user-defined functions.	Program directory
RSW.INI	The ARPEGGIO configuration file; it is optional and can be customized for different users. If RSW.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 RSW.INI setting.	Windows directory
RSWSQL.INI	Translation parameters for ARPEGGIO functions. Distribute this only if you have modified it in some way.	Program directory

Figure 8.1 Report Distribution Files

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 required DLLs and OCXs are included on this list.

Retrieving Report Files

Before ARPEGGIO will display or print a report, it must be able to find all the database tables and other files used in the report, including any text memo file, UDF library file, or image file attached to the report. ARPEGGIO follows the rules described in the following sections to save the locations of database tables and files used in a report, and then to find the tables and files when it retrieves the report. If you are developing reports that will be retrieved from a location other than the one in which they were saved, you should review these rules.

Client/Server Database Platforms

By client/server platforms we mean those platforms for which the tables are physically or logically imbedded in databases, that is those platforms for which there is no simple correspondence between tables and DOS files. Examples of client/server platforms are SQL Server, Oracle, and Netware SQL.

For client/server database platforms, ARPEGGIO will look for tables in the saved data source's database. The data source name saved with a report can be overridden via the Open Report dialog in interactive ARPEGGIO, or via runtime control parameters in the Viewer. Note that standard ODBC behavior calls for a data source named "Default" to be used if it exists and the saved or selected data source does not.

Desktop Database Platforms

ARPEGGIO supports two sets of search rules for certain desktop platforms. In the ODBC style, most emphasis is placed upon the concept of the data-source directory. In the non-ODBC style, data-source directories play a part, but the rules are also based on the locations of files relative to the report library and master table.

ODBC-style search rules are *always* used for desktop platforms other than Xbase or Paradox, and are used with Xbase and Paradox when the setting UseCommonDlg=0 is present in the [Preferences] section of RSW.INI, or if the UseCommonDlg setting is absent entirely. When UseCommonDlg=0 or is absent, ARPEGGIO mimics the client/server method of table selection by presenting a list of tables associated with a given data source. However, if the JoinAcrossDir setting for a given

driver/platform is 1, the table-selection dialog will allow you to select from lists of tables corresponding to directories other than the default one for any data source using that driver.

The use of non-ODBC-style search rules applies only to Xbase and Paradox and even then only if the setting UseCommonDlg=1 is present in the [Preferences] section of RSW.INI. When UseCommonDlg=1, ARPEGGIO uses the ordinary Windows common file dialog for selecting Xbase and Paradox data files.

Alternatively, you can make ARPEGGIO strictly follow client/server behavior for a given desktop driver/platform by setting JoinAcrossDir for the driver/platform in RSWSQL.INI to 0. For Xbase and Paradox you must also set UseCommonDlg to 0 in RSW.INI.

When retrieving a report, ARPEGGIO uses the style of search rules implied by the UseCommonDlg setting in effect at the time the report was created or modified. However, if you resave a report that was originally saved in one style after changing to the other style, the report will be saved in the then-current style.

ODBC Style

In ODBC-Style reports, the search rules for the master table and for related tables are as follows.

Master Table

If the master table was chosen from the default directory of its data source and that data source still exists, the following search rules are used:

A. ARPEGGIO looks in the current default directory of the data source.

B. ARPEGGIO looks in the directory from which the table was originally selected.

C. ARPEGGIO looks in the default data directory specified in RSW.INI.

If the master table was *not* in the default directory of its data source and that data source still exists, the following search rules are used:

A. ARPEGGIO looks in the directory from which the table was originally selected.

B. ARPEGGIO looks in the current default directory of the data source.

C. ARPEGGIO looks in the default data directory specified in RSW.INI.

If the data source from which the master table was chosen does *not* still exist, you will be unable to retrieve the report without using an alternate data source.

Related Tables

If a related table was chosen from the default directory of its data source and that data source still exists, the following search rules are used:

A. ARPEGGIO looks in the current default directory of the data source.

B. ARPEGGIO looks in the directory from which the table was originally selected.

C. ARPEGGIO looks in the current default directory of the data source containing the master table.

D. ARPEGGIO looks the default data directory specified in RSW.INI.

If a related table was *not* chosen from the default directory of its data source and that data source still exists, the following search rules are used:

A. ARPEGGIO looks in the directory from which the table was originally chosen.

B. ARPEGGIO looks in the current default directory of the data source.

C. ARPEGGIO looks in the current default directory of the data source containing the master table.

D. ARPEGGIO looks the default data directory specified in RSW.INI.

If the data source from which the related table was chosen does *not* still exist, ARPEGGIO will allow you to select an alternate data source.

Non-ODBC Style

In the following rules, “data source” directory is the default directory associated with a data source; “master” drive/directory is the drive/directory in which 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 file name of any file as it was when the report was last saved.

Master Table

ARPEGGIO looks in the following places for the master table. Note that some of the search rules are provisional — they do not apply in all cases. If a particular search rule does not apply in retrieving the current report, ARPEGGIO simply goes on to the next search rule. Of course, once ARPEGGIO locates the master table via one of the search rules, it stops applying them; it does not check to see if the master table also exists in other directories.

A. If the master table was in the data-source directory when the report was saved, ARPEGGIO begins by searching in the current data-source directory.

B. If the master table was in the directory containing the report library, ARPEGGIO looks in the drive/directory in which the report library is currently located.

C. ARPEGGIO looks in the saved drive/saved directory of the master table.

D. If the master table was *not* in the data-source directory when the report was saved, ARPEGGIO looks in the current data-source directory.

E. ARPEGGIO looks in the default drive/default directory.

Related Tables

When you save a report, ARPEGGIO follows a set of rules to save the name of each related table. In the following rules, assume that the master table is in C:\DIR1.

A. If the related table was in the data-source directory when the report was saved, ARPEGGIO looks in the current data-source directory.

B1. If the file's drive and directory are the same as the master drive and directory, ARPEGGIO saves only the file name in the report definition. For example, if the full path and name of a related table is C:\DIR1\TABLE1, ARPEGGIO saves only TABLE1.

When the report is retrieved, ARPEGGIO tries to locate required files by searching:

C1. master drive/master directory/saved file name
D1. data-source directory, if not already searched in rule 1
E1. default drive/default directory/saved file name.

B2. If the file's drive is the same as the master drive, but the directories differ, ARPEGGIO saves both the file name and the directory. For example, if a report uses a table in C:\DIR2 and the file name is TABLE2, ARPEGGIO saves \DIR2\TABLE2.

When the report is retrieved, ARPEGGIO tries to locate required files by searching:

C2. master drive/saved directory/saved file name
D2. master drive/master directory/saved file name
E2. data-source directory, if not already searched in rule 1
F2. default drive/default directory/saved file name

B3. If the file's drive differs from the master drive, ARPEGGIO saves the entire path and file name. For example, if the report uses a table whose name is D:\DIR3\TABLE3, ARPEGGIO saves D:\DIR3\TABLE3.

When the report is retrieved, ARPEGGIO tries to locate required files by searching:

C3. saved drive/saved directory/saved file name
D3. master drive/master directory/saved file name
E3. data-source directory, if not already search in rule 1
F3. default drive/default directory/saved file name.

Text Memo Files

When you save a report, ARPEGGIO saves the complete path and name of any attached text memo file. When you retrieve a report that uses such a file, ARPEGGIO first looks for the file in the drive/directory saved with the report. It next looks in the default data directory specified in RSW.INI.

Image Files

When you save a report, Report Designer records whether the image was saved in the same directory as the report. If it was, the image file is searched for in the directory that contains the current report. If the image was not saved in the same directory as the report, the image file is searched for in the directory that contained it when the report was saved. If Report Designer does not find the image in the report directory or the saved directory, it will look in the default image directory. If the Viewer does not find the image file in the report or saved directory, it will search the default image directory specified in the RSW.INI file or identified with the **/I** switch.

Consistency Checking

When a report is retrieved, ARPEGGIO checks to see whether the saved report is consistent with the current table definition. ARPEGGIO will notify you of discrepancies between the report and the tables it uses. For example, ARPEGGIO checks to see whether you have changed the column names in any of the tables since the last time you saved the report.

The following database changes affect reports saved in ARPEGGIO:

- Deleting a column;
- Changing the name or data type of a column;
- Changing the width of a column;
- Changing the name of a table.

The following sections explain how ARPEGGIO responds to the changes described above. In many cases, ARPEGGIO notifies you of the inconsistency between the report and the database. You can then edit the retrieved report in interactive ARPEGGIO to accommodate the changes made to your tables. (Note that you will not be notified of changes in column width.)

Deleted Columns

If you delete from a table any column that is used by a report, ARPEGGIO notifies you that the column is missing when it retrieves the report. It erases the column from the report, as well as erasing any totals based on it. If the deleted column 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 ARPEGGIO will print the report.

If you delete a column used in a filter, you will be prompted to edit the filter when you try to print the report.

If you delete a column that is used as a sort or group field, ARPEGGIO 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 sort fields:

1 STATE
2 CITY
3 COMPANY
4 PRODUCT

Your Sort-Group Table will then contain only the following sort fields:

1 STATE
2 CITY

If you deleted columns used to group your report, you may need to edit your total fields so that they reset at the appropriate level.

Deleted Join Column

In order to retrieve the report, ARPEGGIO creates "dummy" join fields that have the same names as the missing join fields, with the prefix ?_ (as in ?_NAME). When ARPEGGIO finishes retrieving the report, it displays the message "Join must be edited." Before printing or previewing a report, use the Database ⇒ Joins dialog to correct or remove each join whose description in the Joins list is flagged with a question mark (?). Any dummy join fields will appear in the dialog prefixed with ?_.

Changed Column Name or Data Type

If you change the name or data type of a column, ARPEGGIO behaves as if the original column had been deleted and a new column added: the original is removed from the composite record structure, along with any fields that total it, and the new one is added to the composite record structure. Use Insert ⇒ Field to insert the new column. Any calculated field that uses the changed column 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 ARPEGGIO will print the report. If you change the name or data type of a column used in a filter or a join, or as a sort or group field, ARPEGGIO will behave as if the column had been deleted, as described above.

Changed Column Width

If you change the width of a column used in the report, ARPEGGIO 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 ⇒ Field to make any desired adjustment. If you use the field more than once in a report, each occurrence must be adjusted individually.

Changed File Name

If a table or file name has been changed, ARPEGGIO uses the search rules previously described. If it can't find the file under its old name, ARPEGGIO then displays the path and name of the file it can't find. Follow the prompts to select or enter the path and file name of the renamed file.

Note: Save the report after correcting for database or file location changes; otherwise, you will have to repeat the corrections the next time you retrieve the report.

Appendix A
Viewer Equivalencies

Introduction

ARPEGGIO provides three ways of accessing the Viewer: the Viewer executable, the Viewer DLL, and the ARPEGGIO 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 Equivalencies

Custom Control Property	*DLL Equivalent*	*Viewer EXE Equivalent*	*Default Value*
(About)			
Action	execRuntime	(Not applicable)	(Not applicable)
CopiesToPrinter	setCopies	RI_COPIES	Saved number
Database	setDatabase	RI_DB	Saved database
DataDirectory	setDataDir	/D	Value in RSW.INI
DataSource	setDataSource	RI_DSOURCE	Saved data source
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)
GroupFields	setGroupField	RI_GROUP1, ...	Saved group fields
ImageDirectory	setImageDir	/I	Value in RSW.INI
Include	setFilterUsage	RI_INCLUDE	Use saved query
LastErrorCode	returned from execViewer	RO_ECODE	(Not applicable)
LastErrorPage	returned from execViewer	RO_PAGES	(Not applicable)
LastErrorString	returned from execViewer	RO_EMSG	(Not applicable)
MasterTable	setMasterTableName	RI_MASTER	Saved value

Figure A.1 Viewer Equivalencies

Custom Control Property	*DLL Equivalent*	*Viewer EXE Equivalent*	*Default Value*
MemoFileName	setMemoName	RI_MEMO	Saved value
NoEscape	setPreventEscape	RI_NOESC	False
Parameters	setUserParam	User-defined parameters	Blank
Password	setPassword	/P	(Not applicable)
Port	setPrinterPort	RI_WPORT	Saved printer port
Printer	setPrinter	RI_WPTR	Saved printer driver
PrintFileName	setOutputFile	RI_OUTFILE	Saved value
RelatedTables	setJoinInfo	RI_ALIAS1, ...	Saved related tables
Replace	setReplace	RI_REPLACE	(Not applicable)
ReportDirectory	setLibraryDir	/R	Value in RSW.INI
ReportLibrary	setLibrary	RI_LIBRARY	Required
ReportName	chooseReport	RI_REPORT	Required
ReportPick	setReportPick	RI_REPPICK	(Not applicable)
SortFields	setSortField	RI_SORT1, ...	Saved sort fields
StartPage	setBeginPage	RI_BEGPAGE	Saved starting page
(Not applicable)	setStatusEveryPage	RI_CHKTIME	R
StatusFileName	setStatusFileName	/O	RSWRUN.OUT
SuppressTitle	setSuppressTitle	/H	False
TestPattern	setTestPattern	RI_TEST	False
UserName	setUserName	/U	(Not applicable)
Where	setWhere	RI_WHERE	(Not applicable)
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

Figure A.1 Viewer Equivalencies (continued)

Index

A

About, 129
Action, 129
Action routines, 46
Asynchronous printing, 120
Auto-SQL reports
  inserting WHERE clauses, 29, 171

B

Btrieve
  default data directory, 9

C

C programs
  calling Viewer from, 36
Case sensitivity, 16
  control table parameters, 14
CDLL( ) function, 195
chooseDataSource, 51
choosePrinter, 52
chooseReport, 53
chooseTable, 55
Command files, 6
Command switches, 6
  /AL, 11
  /B, 10
  /CD, 8
  /CP, 8
  /CS, 8
  /CU, 8
  /D, 9
  /H, 10
  /I, 10
  /O, 10, 107
  /P, 9
  /R, 9
  /T, 5, 7
  /U, 9
Control parameters
  case of, 16
Control tables
  case sensitivity, 14
  errors in, 35
  parameter widths, 14
  parameters for, 13
  predefined parameters, 15
  question mark field values, 110
  required parameters for, 13
  row ID numbers, 5, 20
  specifying password, 8
  specifying user name, 8
  structure of, 12
  user-defined parameters, 40
Control tables and files
  parameters for, 13
  predefined parameters, 15
  required parameters for, 13
  specifying with /T switch, 7
Copies, 17
CopiesToPrinter, 130

D

Database, 131
Database property page, 128
DataDir, 11
DataDirectory, 131
DataSource, 132
dBASE
  default data directory, 9
DBF export, 133
Default data directory
  specifying with /D switch, 9
Default image directory
  specifying with /I switch, 10
Default library directory
  specifying with /R switch, 9
Defaults property page, 128
DEFINE REPORTVIEW
  overriding, 25, 157
Defining parameters, 40
Destination, 133
DisplayError, 135
Displaying errors, 18

Displaying reports, 24, 98, 134
DisplayStatus, 136
DLLs
  calling functions from, 195
Dynamic-link libraries
  calling functions from, 195

E

EndPage, 136
endReport, 56
Error messages, 35
  displaying on screen, 18
Error-Handling routines, 51
Excel Chart, 134
Excel PivotTable, 134
EXEC
  overriding, 25, 157
execRuntime, 57
ExportDestination, 137
Exporting reports to text files, 23, 24, 98

F

Field width
  in control table, 14
Filter, 138
Filter property page, 127
Filters, 18, 138
  and Viewer, 18, 20, 138, 143
  overriding, 20, 143
Forms
  printing on, 29
Functions
  calling from DLLs, 195

G

General property page, 124
getBeginPage, 58
getCopies, 59
getDataSource, 60
getDisplayErrors, 60
getDisplayStatus, 61
getEndPage, 61
getErrorInfo, 62
getExportDest, 64
getFilter, 64
getFilterUsage, 65
getFirstFieldName, 65
getFirstFilteredFieldName, 66
getFirstGroupField, 67
getFirstJoinInfo, 68
getFirstReplace, 69
getFirstSortField, 70
getFirstUserParam, 70
getLibrary, 71
getMasterTableName, 72
getMemoName, 72
getNewReportHandle, 73
getNextFieldName, 73
getNextFilteredFieldName, 74
getNextGroupField, 75
getNextJoinInfo, 75
getNextReplace, 76
getNextSortField, 77
getNextUserParam, 78
getOutputDest, 78
getOutputFile, 79
Get-parameter routines, 46
getPreventEscape, 79
getPrinter, 80
getPrinterPort, 81
getReportPick, 81
getRuntimeRecord, 82
getStatusEveryPage, 83
getTestPattern, 84
getWinTitle, 84
Group property page, 127
GroupFields, 140
GroupFieldsString, 141
Grouping
  overriding with Viewer, 19

I

Image files
  directory location of, 10
ImageDirectory, 142
ImgDir, 11
ImgExt, 11
Include, 142
IndExt, 11
Interrupting reports, 23, 149

J
Joins property page, 128
L
LastErrorCode, 144
LastErrorPage, 145
LastErrorString, 145
LibDir, 11
LoadProperties, 146
M
MasterTable, 147
MemExt, 11
MemoFileName, 148
N
NoEscape, 149
Number of copies, 17
O
Object Linking and Embedding Control;Custom control, 119
OCX, 119
P
Page numbers
  ending, 16, 137, 167
  starting, 16, 137, 167
Parameter passing, 109, 110, 111
  defining parameters, 40
  incorporating values in reports, 41
  question mark parameter value, 40, 110
  RIPARAM, 41
  using control table or file, 39
  using parameter table, 42
Parameter tables, 42
Parameters, 150
Parameters property page, 129
ParametersString, 150
Password, 151
PointSize, 190
Port, 152
PowerBuilder
  calling Viewer from, 38
Preview Window property page, 126
Previewing reports, 24, 98, 134
PrevWinClr, 11
Print To property page, 124
Printer, 153
  overriding, 30, 101, 153
  selecting, 30, 101, 153
Printer port
  overriding, 30, 102, 152
  selecting, 30, 102, 152
Printer property page, 125
PrintFileName, 154
Printing
  on forms, 29
  selected pages, 16, 137, 167
  test patterns, 29
  to file, 23, 99
Prompting users for input, 40, 110
Q
Question mark (?) parameter value, 40
Question mark parameter value, 14, 20, 23, 24, 27, 30, 110
R
Related tables
  overriding, 16
RelatedTables, 155
RelatedTablesString, 155
Replace, 156
Report libraries
  directory location of, 9
  specifying for Viewer, 21
Report Shortcut Maker utility, 4
Report title, 31, 176
ReportDirectory, 159
ReportLibrary, 160
ReportName, 161
ReportPick, 162
Reports
  specifying for Viewer, 161
ResetControl, 163
resetErrorInfo, 85
ResetProperties, 164

RI_ALIAS#, 16
RI_BEGPAGE, 16
RI_CHKTIME, 17
RI_COPIES, 17
RI_DB, 17
RI_DISPERR, 18, 33
RI_DSOURCE, 18
RI_ENDPAGE, 16
RI_EXPDST, 18
RI_FILTER, 18, 19
RI_GROUP, 19
RI_ID, 5, 20, 36
RI_INCLUDE, 20, 143
RI_LIBRARY, 21
RI_MASTER, 22
RI_MEMO, 22
RI_NOESC, 23
RI_OUTFILE, 23
RI_PRINTER, 24
RI_REPLACE, 25
RI_REPORT, 27
RI_REPPICK, 27
RI_SORT, 28
RI_STATUS, 28
RI_TEST, 28
RI_WBORDER, 32
RI_WCTRL, 32
RI_WHEIGHT, 33
RI_WHERE, 29
RI_WLEFT, 33
RI_WMAX, 33
RI_WMIN, 33
RI_WPORT, 30
RI_WPTR, 30
RI_WTITLE, 24, 31, 98, 111
RI_WTOP, 33
RI_WWIDTH, 33
RIPARAM, 41
RO_ECODE, 35
RO_EMSG, 35
RO_PAGES, 35
RO_REPORTS, 35
RO_RIRECNO, 36
RSDECL.BAS, 51
RSREPORT.H, 51
RSW.INI, 11, 121
RSW32.OCX, 119
RSWRUN.EXE, 2, 4
RSWRUN.OUT, 10, 17, 34, 107
RTF, 134
RTF export, 133
RunReport, 164

S

Sample applications
  for OCX, 120
  for the DLL, 45
Saving reports as files, 23, 99
SELECT
  overriding, 25, 157
Selecting
  ending page number, 16, 137, 167
  number of copies, 17
  report library, 21
  starting page number, 16, 137, 167
setBeginPage, 85
setCopies, 86
setDatabase, 86
setDataDir, 87
setDataSource, 87
setDisplayErrors, 88
setDisplayStatus, 88
setEndPage, 89
setExportDest, 90
setFilter, 90
setFilterUsage, 91
setGroupField, 92
setImageDir, 93
setJoinInfo, 94
setLibrary, 94
setLibraryDir, 95
setMasterTableName, 96
setMemoName, 96
setOutputDest, 97
setOutputFile, 99
Set-Parameter routines, 48
setPassword, 100
setPreventEscape, 100
setPrinter, 101

setPrinterPort, 102
setReplace, 103
setReportPick, 104
setSortField, 105
setStatusEveryPage, 106
setStatusFileName, 106
setSuppressTitle, 107
setTestPattern, 107
setUserName, 108
setUserParam, 109
setWhere, 111
setWinBorderStyle, 112
setWinControlBox, 113
setWinHeight, 113
setWinLeft, 114
setWinMaxButton, 114
setWinMinButton, 115
setWinTitle, 115
setWinTop, 116
setWinWidth, 117
ShowSplash, 11
Sort property page, 127
SortFields, 165
SortFieldsString, 166
Sorting
  overriding with Viewer, 28
Specifying Viewer reports, 161
StartPage, 167
Status table
  naming, 107
  specifying alternate name for, 107
StatusFileName, 167
SuppressTitle, 168
Synchronous printing, 120

T

Test patterns
  printing, 29
TestPattern, 169
Text control files, 5
  errors in, 35
  predefined parameters, 15
  required parameters for, 13
  structure of, 12
  user-defined parameters, 40
Text Data, 134
Text Data export, 134
Text export, 133
Text files
  exporting reports to, 23, 24, 98
Text memo files, 22, 148
  directory location of, 9
  overriding, 22, 148

U

UpdateControl, 169
User-Interface routines, 50
UserName, 170
User-SQL property page, 127
User-SQL reports
  overriding SELECT, 25, 157

V

Viewer
  and filters, 20, 143
  command files, 6
  command line for, 4
  command switches, 6
  control parameters, 13
  control tables, 14, 25
  executing multiple reports, 6
  executing selected reports, 5, 20
  executing with a control table, 4
  executing with a text file, 5
  required parameters for, 13
  specifying defaults with RSW.INI, 11, 121
  system requirements for, 2
Viewer output file, 17
Viewer overrides
  filters, 138
  for filters, 18
  for related tables, 16
  Grouping, 19
  master table, 22
  output destination, 24
  output file, 23
  printer, 30, 101, 153
  printer port, 30, 102, 152
  report output file, 99
  saved filter, 20, 143

SELECT, 25, 157
Sorting, 28
text memo files, 22, 148
WHERE clause, 29, 171
Viewer status file, 17, 34
entries, 34
error codes, 35
error messages, 35
format of, 34
naming, 10
number of pages, 35
number of reports, 35
row ID numbers, 36
specifying with /O switch, 10
updating, 17
Viewer status window
cancel button, 23, 149
displaying, 28, 136
Visual Basic
calling Viewer from, 37

W

Where, 170
WHERE clauses
inserting, 29, 171
WindowBorderStyle, 172
WindowControlBox, 173
WindowHeight, 173
WindowLeft, 174
WindowMaxButton, 174
WindowMinButton, 175
WindowTitle, 176
WindowTop, 176
WindowWidth, 177
WinExec, 37
Word Merge, 134
Word Merge export, 134
Worksheet export, 133
writeRuntimeRecord, 117