Chapter 3
Accessing the Runtime DLL

Introduction

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

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

- ❑ Initiate an "instance" of the runtime DLL for use by your application with **initRuntimeInstance**. Then repeat the following four steps as many times as necessary.
- ❑ Select a report/library combination with **chooseReport** or **getRuntimeRecord**. Or, select a report library with **getNewReportHandle** and **setLibrary**.
- ❑ Use various routines to get and set runtime control parameters.
- ❑ Use **writeRuntimeRecord** to save the parameters in a Runtime Control File for later execution or **execRuntime** to use the R&R Runtime to run the report immediately.
- ❑ Clean up the current report with **endReport**.
- ❑ When your application has completed its last report, clean up your "instance" of the runtime DLL with **endRuntimeInstance**.

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

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

Action Routines

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

- **chooseReport** specifies a library/report combination.
- **endReport** cleans up resources associated with a given report.
- **endRuntimeInstance** frees resources associated with an instance of the runtime DLL.
- **execRuntime** runs a given report.
- **getNewReportHandle** obtains the handle of an empty report-information structure.
- **getRuntimeRecord** specifies a library/report combination along with parameter values as defined in a Runtime Control File record.
- **initRuntime Instance** initializes an instance of the runtime DLL and must be called before any other routines in this API.
- **writeRuntimeRecord** writes to a Runtime 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 Runtime 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 library. 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 library unless the parameter is overridden in the runtime control file record, in which case the current value is the value from the control file record. Once you have used a set-parameter routine for the parameter, the current value is the value you specified via the set-parameter routine.

- ❑ If you have initiated the processing of a report via a call to **getNewReportHandle** and have not yet used a set-parameter routine for a given parameter, the current value of the parameter is the default value of that parameter. Once you have used a set-parameter routine for the parameter, the current value is the value you specified via the set-parameter routine.

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

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

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

Set-Parameter Routines

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

- ❑ **setBeginPage** sets the value of the starting-page-number parameter.
- ❑ **setCopies** sets the value of the number-of-copies parameter.

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

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

User-Interface Routines

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

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

Error-Handling Routines

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

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

Functions Provided by the Runtime DLL

The following sections present detailed descriptions of the functions provided by the Runtime 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 Runtime DLL is defined in two header files, one named RREPORT.H, for use in C/C++ programs, and one named RRDECL.BAS for use in Visual Basic programs.

choosePrinter

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

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

Return Value

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

Description

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

Related Functions

setPrinter, setPrinterPort, getPrinter, getPrinterPort

Example

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

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

chooseReport

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

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

Return Value

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

Description

The *lpszLibName* argument must point to the name of a report library. If *lpszLibName* does not include a path, the Runtime looks for the library in the default library directory specified in RRW.INI. If no default is specified in the INI file either, the Runtime looks for the library in the current directory.

The *lpszRepName* argument points to a buffer containing either a null string or the name of a report. If *lpszRepName* contains the name of a report, **chooseReport** opens the library specified by *lpszLibName*, reads the report specified by *lpszRepName*, and prepares a report-information structure based on that report. If *lpszRepName* points to a null string, **chooseReport** presents a dialog via which the user can select a report from the list of reports available in the specified library. After the user selects a report, **chooseReport** opens the specified library, reads the selected report, copies the report name into *lpszRepName* to the extent allowed by *size*, and prepares a report-information structure based on that report. The handle returned by **chooseReport** is used as input to most other functions contained within this API.

The *lpszAppName* argument identifies the calling application.

Related Functions

endReport, getRuntimeRecord

Example

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

```
{
   char repname[40];
   int hRpt;

   hRpt = chooseReport ((LPSTR)"Application Name",
      (LPSTR)"c:\\libs\\reports.rp5", (LPSTR)repname, 40);
}
```

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, endRuntimeInstance

Example

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

```
endReport (hRpt);
```

endRuntimeInstance

BOOL FAR PASCAL endRuntimeInstance (void);

Return Value

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

Description

Call the **endRuntimeInstance** function to signify that your application is finished with the runtime DLL. This enables **endRuntimeInstance** to clean up resources associated with this instance of the runtime DLL.

Related Functions

initRuntimeInstance, endReport

Example

To inform the DLL that you are finished with it:

```
endRuntimeInstance();
```

execRuntime

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

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

Return Value

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

Description

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

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

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

Regardless of the value of *bWait*, any error 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 runtime 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 Runtime will create a runtime status file and the information provided by *lpiECode*, *lpiPageCount*, and *lpiEMsg* is instead provided by the fields RO_ECODE, RO_PAGES, and RO_EMSG. See Chapter 2 for details of the runtime status file.

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

Related Functions

chooseReport, getRuntimeRecord, getNewReportHandle, setBeginPage, setEndPage, setStatusEveryPage

Example

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

```
{
   int ecode;
   long pages;
   char emsg[200];
   int done = FALSE;

   while (!done)
   {
      // code to let user make changes to parameters, etc.
      execRuntime (hRpt,    // report handle
                   1,       // synchronous
                   SW_SHOW,   // current size/position
                   (LPINT)&ecode,         // place for error code
                   (LPLONG)&pages,  // ... pages printed
```

```
                      (LPSTR)emsg,     // ... error message
                      200);    // size of emsg buffer
         switch (ecode)
         {
            case 'N':     // success
            case 'C':     // user canceled report
               done = 1;  // either way, we're happy
               break;
            case 'J':     // problem with parameters
               // error handling code
               break;
            case 'R':     // problem running report
               // error handling code
               break;
         } // end switch
      } // end while
   }
```

getBeginPage

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

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

Return Value

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

Description

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

Related Functions

setBeginPage, getEndPage, setEndPage, execRuntime

Example

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

```
{
   LONG begPage;

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

getCopies

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

hReport	Report handle.

lpiCopies	Number-of-copies buffer.

Return Value

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

Description

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

Related Functions

setCopies

Example

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

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

getDisplayErrors

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

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

Return Value

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

Description

Use **getDisplayErrors** to obtain the current value of the "display errors" parameter for the report specified by *hReport*. **getDisplayErrors** returns the parameter in the form of a boolean in the buffer pointed to by *lpbDispErr*. See **setDisplayErrors** for a discussion of this parameter.

Related Functions

setDisplayErrors

Example

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

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

getDisplayStatus

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

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

Return Value

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

Description

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

Related Functions

setDisplayStatus, **getPreventEscape**, **setPreventEscape**

Example

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

```
{
   BOOL dispStatus;

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

getEndPage

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

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

Return Value

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

Description

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

Related Functions

setEndPage, getBeginPage, setBeginPage

Example

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

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

getErrorInfo

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

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 this instance of the runtime DLL. 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 Runtime 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 Runtime Control File specified as *lpszControlFile* to **getRuntimeRecord**.
- ❑ **L** (Library) indicates a problem with a report library being processed by the Runtime DLL. It would be returned, for example, if **chooseReport** were unable to read the report library specified as *lpszLibName*.
- ❑ **S** (Syntax) indicates a problem with the arguments passed to the routine generating the error. This might indicate NULL passed for a required pointer or a buffer size of zero, for example.
- ❑ **V** (Value) indicates that no value is available for the parameter whose value you have requested.

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

Related Functions

resetErrorInfo

Example

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

getExportDest

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

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

Return Value

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

Description

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

Related Functions

setExportDest

Example

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

```
{
   char dest[2];
```

```
      getExportDest (hRpt, (LPSTR)dest);
   }
```

getFilter

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

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

Return Value

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

Description

Use **getFilter** to obtain the current value of the "filter" parameter for the report specified by *hReport*. **getFilter** returns the current filter (in the form of a valid R&R 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 R&R or overridden in the Runtime Control File record if *hReport* was obtained via a call to **getRuntimeRecord**. If **setFilter** has been called to override the filter saved with the report, **getFilter** simply returns the value previously set. See **setFilter** for details of filter expressions. See **setFilterUsage** for details of the interaction between values set by **setFilterUsage** and **setFilter**.

Related Functions

setFilter, getFilterUsage, setFilterUsage

Example

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

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

getFilterUsage

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

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

Return Value

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

Description

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

Related Functions

setFilterUsage, getFilter, setFilter

Example

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

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

getFirstFieldName

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

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

Return Value

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

Description

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

Related Functions

getFirstFilteredFieldName, getNextFieldName

Example

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

```
int InitFieldCombo(HWND hCombo, int hRpt)
{
   char szField[80];     // buffer for field name
   int nFields = 1;   // return count of fields

   // Extract field names from the report.
   if (getFirstFieldName(hRpt, szField, sizeof(szField)))
```

```
        {
           ComboBox_AddString(hCombo, szField);
           while (getNextFieldName(hRpt, szField, sizeof(szField)))
           {
              ComboBox_AddString(hCombo, szField);
              // This returns false if not an iterator error.
              if (!getError())
                 return FALSE;
              nFields++;
           }
        }
        else return getError();  // Error handling routine.
        return nFields;
     }
```

getFirstFilteredFieldName

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

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

Return Value

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

Description

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

The *filter* argument specifies the context to be used in deciding which available fields to return. The valid values for *filter*, defined in rreport.h, are FILTER_ID_SORT 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);
}
```

getFirstRelationInfo

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

hReport	Report handle.

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

Return Value

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

Description

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

Related Functions

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

Example

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

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

getFirstSortField

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

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

Return Value

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

Description

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

Related Functions

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

Example

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

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

getFirstUserParam

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

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

Return Value

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

Description

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

Related Functions

getNextUserParam, **setUserParam**

Example

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

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

getHighScope

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

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

Return Value

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

Description

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

Related Functions

setHighScope, **getLowScope**, **setLowScope**, **getScopeUsage**, **setScopeUsage**

Example

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

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

getLibrary

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

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

Return Value

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

Description

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

Related Functions

setLibrary

Example

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

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

getLowScope

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

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

Return Value

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

Description

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

Related Functions

setLowScope, **getHighScope**, **setHighScope**, **getScopeUsage**, **setScopeUsage**

Example

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

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

getMasterIndexInfo

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

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

Return Value

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

Description

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

Related Functions

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

Example

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

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

getMasterTableName

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

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

Return Value

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

Description

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

Related Functions

setMasterTableName

Example

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

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

getMemoName

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

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

Return Value

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

Description

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

Related Functions

setMemoName

Example

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

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

getNewReportHandle

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

lpszAppName	Name of calling application.

Return Value

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

Description

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

Related Functions

chooseReport, getRuntimeRecord, setReportPick

Example

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

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

getNextFieldName

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

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

Return Value

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

Description

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

Related Functions

getFirstFieldName, getNextFilteredFieldName

Example

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

getNextFilteredFieldName

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

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

Return Value

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

Description

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

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

Related Functions

getFirstFilteredFieldName, **getFirstFieldName**, **getNextFieldName**

Example

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

getNextGroupField

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

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

Return Value

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

Description

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

Related Functions

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

Example

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

getNextRelationInfo

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

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

Return Value

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

Description

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

Related Functions

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

Example

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

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

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

getNextSortField

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

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

Return Value

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

Description

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

Related Functions

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

Example

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

getNextUserParam

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

hReport	Report handle.
lpszName	Parameter-name buffer.

nSize	Size of *lpszName* buffer.
lpszValue	Parameter-value buffer.
vSize	Size of *lpszValue* buffer.

Return Value

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

Description

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

Related Functions

getFirstUserParam, **setUserParam**

Example

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

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

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

getOutputDest

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

hReport	Report handle.
lpszDest	Output-destination 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*. See **setOutputDest** for a discussion of this parameter.

Related Functions

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

Example

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

```
{
   char dest[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***);**

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

Return Value

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

Description

Use **getRuntimeRecord** to begin processing a report based on information in the ASCII Runtime Control File whose name is pointed to by *lpszControlFile*. The *lpszAppName* argument identifies the calling application. The handle returned by **getRuntimeRecord** is used as input to most other functions contained within this API.

Related Functions

chooseReport, getNewReportHandle, writeRuntimeRecord

Example

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

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

getScopeUsage

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

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

Return Value

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

Description

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

Related Functions

setScopeUsage, getLowScope, setLowScope, getHighScope, setHighScope

Example

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

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

getStatusEveryPage

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

hReport	Report handle.
lpbStatus	Status-frequency buffer.

Return Value

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

Description

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

Related Functions

setStatusEveryPage

Example

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

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

getTestPattern

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

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

Return Value

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

Description

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

Related Functions

setTestPattern

Example

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

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

getWinTitle

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

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

Return Value

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

Description

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

Related Functions

setWinTitle

Example

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

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

initRuntimeInstance

BOOL FAR PASCAL initRuntimeInstance (void);

Return Value

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

Description

Use **initRuntimeInstance** to initialize an instance of the runtime DLL. Your application must call this routine before calling any other runtime DLL routines. When your application is finished with the runtime DLL, it must call **endRuntimeInstance**.

Related Functions

endRuntimeInstance

Example

To prepare the runtime DLL for further calls:

```
initRuntimeInstance();
```

resetErrorInfo

BOOL FAR PASCAL resetErrorInfo (void);

Return Value

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

Description

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

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 R&R to revert to the number of copies saved with the report.

Related Functions

getCopies

Example

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

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

setDataDir

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

hReport	Report handle.
lpszDir	Default data directory.

Return Value

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

Description

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

Related Functions

setImageDir, setLibraryDir

Example

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

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

setDisplayErrors

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

hReport	Report handle.
bDisperr	Display-errors flag.

Return Value

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

Description

Use **setDisplayErrors** to replace the current value of the “display errors” flag for the report specified by *hReport* with the value specified by *bDisperr*. If the “display errors” flag is non-zero, error messages generated by R&R 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 Runtime 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*, Runtime 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, R&R will display a status window while it is generating the report; otherwise it will display an icon while it is running. By default, R&R 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.

Related Functions

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

Example

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

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

setEndPage

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

hReport	Report handle.
lEndPage	Ending page number.

Return Value

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

Description

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

Related Functions

getEndPage, setBeginPage, getBeginPage

Example

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

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

setExportDest

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

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

Return Value

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

Description

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

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

Related Functions

getExportDest, setOutputFile

Example

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

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

setFilter

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

hReport	Report handle.
lpszFilter	Filter expression.

Return Value

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

Description

Use **setFilter** to specify a filter expression, *lpszFilter*, that may be used instead of the filter, if any, saved with the report specified by *hReport*. R&R 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 R&R uses the expression specified via **setFilter**, it will include only those records where the value of the expression is true. The maximum size of a filter expression is 1024.

Related Functions

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

Example

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

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

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

setFilterUsage

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

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

Return Value

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

Description

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

- ❑ **S** (Saved) means to run the report using the filter saved with it, if any. R&R will ignore any expression specified via **setFilter** and run the report exactly as it was saved.
- ❑ **E** (Entire) means to ignore any filter saved in the report or specified via **setFilter**.
- ❑ **O** (Override) means to override the saved filter, if any, with the expression specified via **setFilter**.
- ❑ **?** (Question mark) means to allow the user to enter a filter or edit the saved filter at runtime. If no filter was saved with the report, the Insert Selection Rule dialog displays, as shown in Figure 3.1.

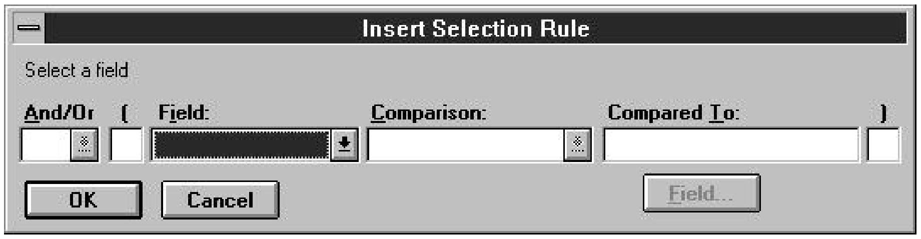
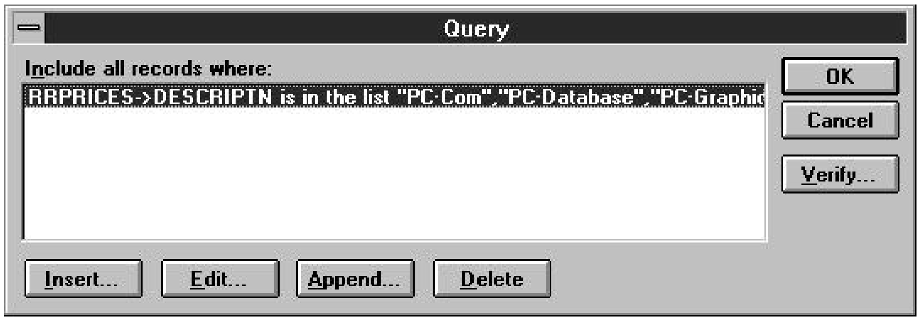


Figure 3.1 Insert Selection Rule Dialog Box

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



RRPRICES->DESCRIPTN is in the list "PC-Com","PC-Database","PC-Graphic

Figure 3.2 Query Dialog Box

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

Related Functions

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

Example

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

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

setGroupField

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

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

Return Value

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

Description

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

Related Functions

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

Example

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

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

setHighScope

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

hReport	Report handle.
lpszScope	High-scope value.

Return Value

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

Description

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

Related Functions

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

Example

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

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

setImageDir

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

hReport	Report handle.
lpszDir	Default image directory.

Return Value

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

Description

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

Related Functions

setDataDir, setLibraryDir

Example

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

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

setIndexExtension

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

hReport	Report handle.
extNumber	Extension index.

Return Value

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

Description

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

The possible values and meanings for *extNumber* are:

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

Related Functions

setMasterIndexInfo, setRelationInfo

Example

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

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

setLibrary

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

hReport	Report handle.
lpszName	Library-name buffer.

Return Value

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

Description

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

If *lpszName* does not include a path, the Runtime looks for the library in the directory specified by **setLibraryDir**. If **setLibraryDir** has not been called, the Runtime looks in the default library directory specified in RRW.INI. If no default is specified in the INI file either, the Runtime 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 RRW.INI with the value specified by *lpszDir*, for the report specified by *hReport*. R&R 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");
```

setLowScope

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

hReport	Report handle.
lpszScope	Low-scope value.

Return Value

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

Description

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

Related Functions

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

Example

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

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

setMasterIndexInfo

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

hReport	Report handle.

lpszPath	Index-name buffer.
cType	Data type.
lpszTag	Tag-name buffer.

Return Value

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

Description

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

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

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

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

Related Functions

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

Example

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

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

setMasterTableName

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

hReport	Report handle.
lpszTable	Name buffer.

Return Value

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

Description

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

Example

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

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

setMemoName

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

hReport	Report handle.
lpszPath	Pathname buffer.

Return Value

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

Description

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

- ❑ If *lpszPath* specifies both a directory and a table name, this directory is the only directory searched and this file name is the only file the Runtime searches for.
- ❑ If *lpszPath* specifies a directory without a file name, the Runtime searches the specified directory for the ASCII memo file name saved with the report.
- ❑ If *lpszPath* specifies a file name without a directory, the Runtime searches for a file with the specified name in the directory of the ASCII memo file saved with the report, then in the default data directory specified via **setDataDir** or in RRW.INI. If no default is specified via **setDataDir**, the Runtime 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**, Runtime will print to the destination saved with the report (or to the printer specified via **setPrinter** function). This parameter can have one of the following values: **D, A, T, P, Excel Chart, Excel PivotTable, CSV, MSWORD, RTF, W, X,** or a question mark (**?**).

- ❑ A value of **D** specifies that the report be sent to the display, allowing the user to preview the report before printing it. After previewing the report, the user can select Print on the Preview screen to send the report to the printer saved with the report or specified via the **setPrinter** function. Note that if the value of *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 pivot table, respectively. You can use this in conjunction with **setExportDest** to control the export destination (display, file, or printer).
- ❑ A value of **CSV, MSWORD,** or **RTF** specifies that the report be exported to a text data file, Word Merge file, or Rich Text Format file, respectively, using either the saved file name or the file name specified via **setOutputFile**.
- ❑ A value of **W** specifies that the report be exported to a worksheet file whose name is specified via **setOutputFile**.
- ❑ A value of **X** specifies that the report be exported to an Xbase file whose name is specified via **setOutputFile**.

- ❑ A value of question mark (**?**) allows the user to select the print destination (screen or printer) at runtime. When the value of *cDest* is a question mark, the user will see the dialog box shown in Figure 3.3. If a title has been specified via **setWinTitle**, the title bar will contain that title; otherwise, the title bar will contain the report name.

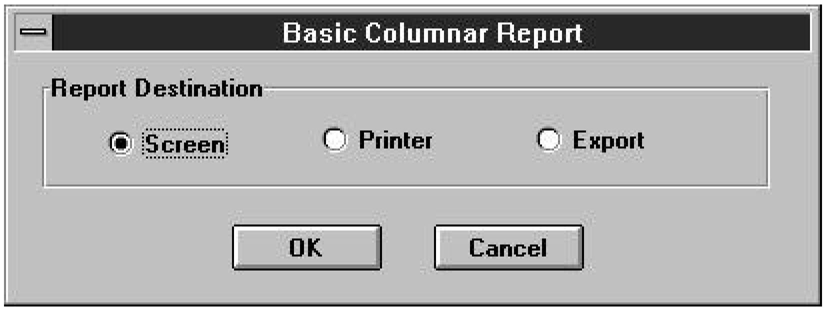


Figure 3.3 Print Destination Dialog Box

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

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

Related Functions

getOutputDest, setOutputFile, getOutputFile, setPrinter, getPrinter

Example

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

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

setOutputFile

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

hReport	Report handle.
lpszName	Output filename.

Return Value

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

Description

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

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

C:\PROJECT\TEXT\INVOICE.TXT

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

Related Functions

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

Example

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

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

setPreventEscape

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

hReport	Report handle.
bNoEsc	Prevent-escape flag.

Return Value

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

Description

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

Related Functions

getPreventEscape, **setDisplayStatus**, **getDisplayStatus**, **execRuntime**

Example

To specify that, for the report whose handle is *hRpt*, Runtime 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.

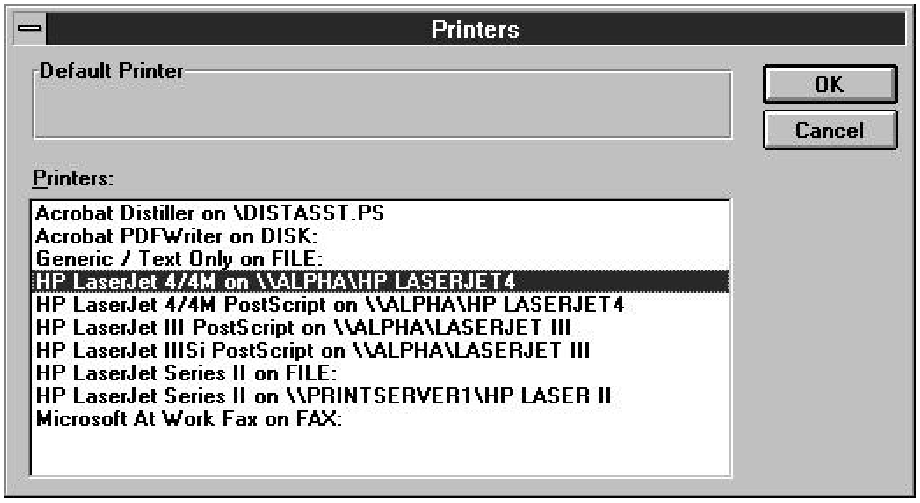
Return Value

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"). Available Windows printers are listed in the R&R Printers dialog (accessed by selecting File ⇒ Printers in R&R). The value is case insensitive (that is, you can enter it in upper, lower, or mixed case).
- The question mark (**?**) value, to allow the user to select a printer at runtime. When the *lpszPrinter* value is a question mark, the Printers dialog will display, as shown in Figure 3.4.



```
Acrobat Distiller on \DISTASST.PS
Acrobat PDFWriter on DISK:
Generic / Text Only on FILE:
HP LaserJet 4/4M on \\ALPHA\HP LASERJET4
HP LaserJet 4/4M PostScript on \\ALPHA\HP LASERJET4
HP LaserJet III PostScript on \\ALPHA\LASERJET III
HP LaserJet IIISi PostScript on \\ALPHA\LASERJET III
HP LaserJet Series II on FILE:
HP LaserJet Series II on \\PRINTSERVER1\HP LASER II
Microsoft At Work Fax on FAX:
```

Figure 3.4 Printers Dialog Box

Printer Setup in the Windows Control Panel controls which printers are listed in the Printers dialog box. R&R initially highlights the printer saved with the report. The user can select another printer and port, or click the Setup button to change other print characteristics.

Related Functions

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

Example

To allow the user to select a printer interactively in Runtime 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 for this parameter. When the value of *lpszPort* is a question mark, the user will see the Printer Setup dialog box shown in Figure 3.4. (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 Runtime for the report whose handle is *hRpt*:

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

setRelationInfo

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

hReport	Report handle.
lpszFilePath	Related file name.

lpszIndexPath	Index file name.
lpszTag	Tag name.
lpszAlias	Alias.
aliasNum	Relation-override number.

Return Value

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

Description

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

Related Functions

getFirstRelationInfo, **getNextRelationInfo**, **setMasterTabl eName**, **getMasterTableName**

Example

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

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

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

setReportPick

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

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

Return Value

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

Description

Use **setReportPick** to replace the current value of the report-selection flag for the report specified by *hReport* to the value specified by *cPickFlag*. If the report-selection flag is set to **R**, the Runtime will prompt the user to select a report from the current report library. If the flag is set to **?**, the Runtime 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 Runtime for the report whose handle is *hRpt*:

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

setScopeUsage

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

hReport	Report handle.
cScopeFlag	Scope-usage flag.

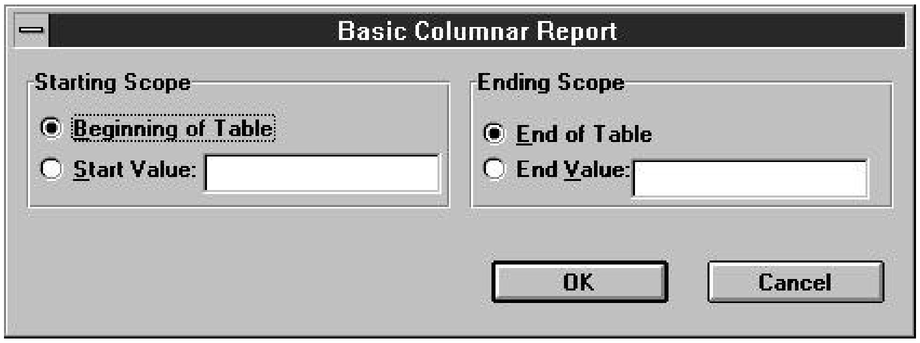
Return Value

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

Description

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

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



Starting Scope: (● Beginning of Table) ○ Start Value:
Ending Scope: (● End of Table) ○ End Value:

Figure 3.5 High and Low Scope Dialog Box

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

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

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

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

Related Functions

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

Example

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

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

setSortField

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

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

Return Value

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

Description

Use **setSortField** to replace an existing sort field or add a new one to the report specified by *hReport*. Pass the sort field number to be added or replaced in *sortNum* and its value in *lpszName*. The *lpszName* argument begins with a **+** or **-** to indicate ascending or descending, respectively, followed by the name of the sort field. You must replace all sort fields from sort field 1 through the last sort field you wish to replace. For example, if you only wish to replace sort field 2, you must call **setSortField** twice, once with a *sortNum* of 1 and once with a *sortNum* of 2. To obtain the current sort field parameters, use **getFirstSortField** and **getNextSortField**.

Related Functions

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

Example

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

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

setStatusEveryPage

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

hReport	Report handle.
bStatus	Status-frequency value.

Return Value

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

Description

Use **setStatusEveryPage** to specify a value for the “status every page” parameter for the report specified by *hReport*. This parameter is meaningful only when **execRuntime** is to be called with a value of zero for *bWait*, in which case R&R will generate a runtime status file. If *bWait* is non-zero, no runtime status file is generated and status is returned to the calling application via **execRuntime**. If *bStatus* is non-zero and R&R 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 runtime status should be written after every page of the report whose handle is *hRpt*:

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

setStatusFileName

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

hReport	Report handle.
lpszPath	Status file name

Return Value

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

Description

Use **setStatusFileName** to specify a status file name for the report specified by *hReport*. A status file is created only if you call **execRuntime** with a *bWait* parameter of 0. You can distinguish Runtime 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 Runtime executable will create a file named RRUNOUT.DBF in the specified directory. If you specify a filename without a path, the specified file will be created in the current directory.

Example

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

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

setSuppressTitle

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

hReport	Report handle.
bValue	Suppress-title flag.

Return Value

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

Description

Use **setSuppressTitle** to set the "suppress title and summary areas" flag for the report specified by *hReport*. If the value of *bValue* is non-zero, R&R will not output Title and Summary areas for reports which contain no records; otherwise R&R 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, R&R will display a dialog allowing the user to print a test pattern before printing the report. The dialog will contain OK, Cancel, and Print buttons. The user can select OK to print a test pattern as many times as necessary to align forms in the printer, and then select Print to print the report. A test pattern includes only page header, record, and page footer lines.

Related Functions

getTestPattern

Example

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

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

setUserParam

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

hReport	Report handle.
lpszName	Parameter-name buffer.
lpszValue	Parameter-value buffer.

Return Value

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

Description

Use **setUserParam** to give the value specified by *lpszValue* to the user parameter whose name is specified by *lpszName* for the report specified by *hReport*.

When the R&R Runtime is called directly using a Runtime Control File, a user parameter is a control-file field that is not defined by R&R. The value of a user parameter is obtained within a report via the R&R function RIPARAM. When the R&R Runtime is called via the Runtime DLL, the DLL deduces the names of the user parameters by searching all calculated fields for uses of the RIPARAM function. The order in which **getFirstUserParam** and **getNextUserParam** return user parameters is not significant. A given user parameter will only have a current value if **setUserParam** has previously been called for that parameter. All user parameters must be of data type character. You can use conversion functions such as CTOD() and VAL() to convert to other data types for use in calculations.

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

Follow these general steps to pass parameters to reports using **setUserParam**.

1. Define calculations in your report using the RIPARAM( ) function.
2. Obtain values for use in the calculations in either of two ways:
   - ❑ Create your own menus or prompts within your application.
   - ❑ Enter a question mark as the value of the control table field.
3. If your application has obtained values for user parameters, pass the values via calls to **setUserParam**; if you wish R&R 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 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. 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 Runtime DLL via **setUserParam**.
- ❑ Enter a question mark (**?**) value for any user-defined field. Whenever a user-defined field contains a question mark, the user will be prompted to enter a value.

Using the Question Mark Field Value

The simplest way to get user input for reports is to use a question mark (**?**) as the value for a user parameter. Optionally, the value can also include the text you want to appear as a prompt. For example, if you want to prompt the user for his or her name, you might define an AUTHOR user parameter and give it the value "?Enter your name:". At runtime, the user will see the dialog shown in Figure 3.6.

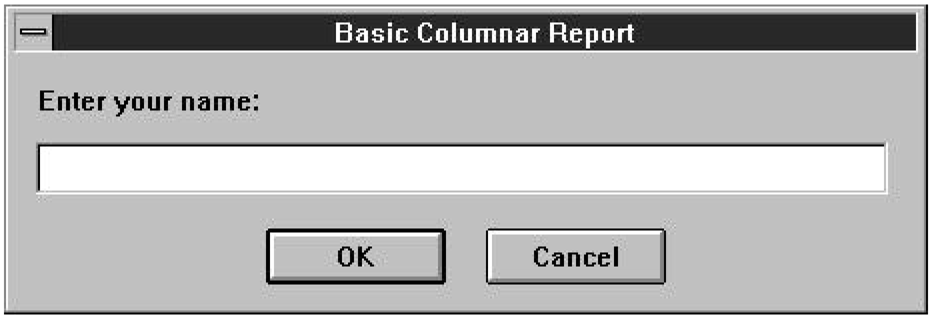


Figure 3.6 Runtime Dialog Box with Prompt

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 Runtime uses the report name. If the user selects the Cancel button, the report will not run and the Runtime will write the "Canceled" message to the status file.

If your control table field contains a question mark only and no text string, the Runtime displays the dialog box shown in Figure 3.6 with the prompt "Enter value for (USER PARAMETER)", as in "Enter value for AUTHOR".

Passing Parameter Values to the Runtime DLL

After obtaining values for user parameters, the final step is to pass those values to the Runtime DLL so they become available for use in RIPARAM() calculations. Use **setUserParam** to specify values for user parameters.

Related Functions

getFirstUserParam, **getNextUserParam**

Example

To specify, for the report whose handle is *hRpt*, a value of R. T. Firefly for the user parameter named AUTHOR:

```
setUserParam (hRpt, (LPSTR)"AUTHOR", (LPSTR)"R. T. Firefly");
```

setWinBorderStyle

BOOL FAR PASCAL setWinBorderStyle (int *hReport*, **int** *style***);**

hReport	Report handle.
style	Preview window border style.

Return Value

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

Description

Use **setWinBorderStyle** to specify the type of border for the preview window for the report specified by *hReport*. The valid values for *style* are:

- ❑ If *style* is 0, the preview window will have no border.
- ❑ If *style* is 1, the preview window will have a fixed size and a single line border.
- ❑ If *style* is 2, the user will be able to change the size of the preview window.
- ❑ If *style* is 3, the preview window will have a fixed size and a double line border.

Related Functions

setWinControlBox, **setWinHeight**, **setWinLeft**, **setWinMaxButton**, **setWinMinButton**, **setWinParentHandle**, **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**, **setWinParentHandle**, **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**, **setWinParentHandle**, **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**, **setWinParentHandle**, **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**, **setWinParentHandle**, **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**, **setWinParentHandle**, **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);
```

setWinParentHandle

BOOL FAR PASCAL setWinParentHandle (int *hReport*, **int** *hParent***);**

hReport	Report handle.
hParent	Window handle of parent window.

Return Value

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

Description

Use **setWinParentHandle** to identify, for the report specified by *hReport*, the window handle of the parent window. Specify the parent window handle via *hParent*. To make the preview window a child of the main Runtime window, specify a value of 0 for *hParent* or simply do not call **setWinParentHandle**.

Related Functions

setWinBorderStyle, **setWinControlBox**, **setWinHeight**, **setWinLeft**, **setWinMaxButton**, **setWinMinButton**, **setWinTitle**, **setWinTop**, **setWinWidth**

Example

To specify that the preview window for the report whose handle is *hRpt* should be a child of the application whose window handle is *hWnd* and should begin 50 pixels down and 40 pixels to the right of the upper-left corner of the screen:

```
setWinParentHandle (hRpt, hWnd);
setWinTop (hRpt, 50);
setWinLeft (hRpt, 40);
```

setWinTitle

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

hReport	Report handle.
lpszTitle	Report title.

Return Value

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

Description

Use **setWinTitle** to set the value of the "report title" parameter for the report specified by *hReport* to the text specified by *lpszTitle*.

The report title is displayed in the following places:

- ❑ The title bar of the Preview window
- ❑ The Print Status window (if **setDisplayStatus** is called with a non-zero *bStatus* value.)
- ❑ Below the Runtime icon (if **setDisplayStatus** is called with a *bStatus* value of zero.)
- ❑ The title bar of the dialog boxes that display if **setPrinter** or **setPrinterPort** is called with an *lpszPrinter* value of question mark, or if **setScopeUsage** is called with an *lpszScopeFlag* value of question mark.

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

Related Functions

getWinTitle, setStatus, setPrinter, setScopeUsage

Example

To specify that the preview window for the report whose handle is *hRpt* should have a title of "on the desktop of Rufus T. Firefly":

```
setWinTitle (hRpt, (LPSTR)"on the desktop of Rufus T. Firefly");
```

setWinTop

BOOL FAR PASCAL setWinTop (int *hReport*, **int** *top***);**

hReport	Report handle.
top	Preview window top-edge position.

Return Value

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

Description

Use **setWinTop** to specify the position of the top edge of the preview window for the report specified by *hReport*. *top* specifies how far, in pixels, from the top edge of the screen the top edge of the preview window is to be.

Related Functions

setWinBorderStyle, **setWinControlBox**, **setWinHeight**, **setWinLeft**, **setWinMaxButton**, **setWinMinButton**, **setWinParentHandle**, **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**, **setWinParentHandle**, **setWinTitle**, **setWinTop**

Example

To specify that the preview window for the report whose handle is *hRpt* should have a single-line border and a fixed size of 400 pixels wide and 300 pixels high:

```
setWinBorderStyle (hRpt, 1);
setWinWidth (hRpt, 400);
setWinHeight (hRpt, 300);
```

setWriteAllow

BOOL FAR PASCAL setWriteAllow (int *hReport*, **BOOL** *bAllow***);**

hReport	Report handle.
bAllow	Allow-write flag.

Return Value

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

Description

Use **setWriteAllow** to set the “allow write” flag for the report specified by *hReport*. If the value of *bAllow* is non-zero, R&R will open database and index files in a mode that allows them to be modified while R&R is using them. The allow-write flag specified via **setWriteAllow** overrides the flag value specified in RRW.INI. Even if the allow-write flag is set to false, other R&R users will be able to report on files you are using with R&R.

For more information about file write access settings, see the explanation of the Allow Other Users to Update Database Tables setting in Chapter 6, "Setting Defaults," in *Using R&R*. Note that this switch controls R&R's behavior only when accessing shared data; your database or network software may impose other file access restrictions.

Related Functions

none

Example

To specify that R&R should allow other users to modify databases and indexes it is using to generate the report specified by *hRpt*:

```
setWriteAllow (hRpt, 1);
```

setXbaseEditor

BOOL FAR PASCAL setXbaseEditor (int *hReport*, **BOOL** *bXbase***);**

hReport	Report handle.
bXbase	Xbase-memos flag.

Return Value

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

Description

Use **setXbaseEditor** to set the "Xbase memos" flag for the report specified by *hReport*. If the value of *bXbase* is non-zero, R&R will assume that all database memos in use by this report were created with an Xbase memo editor.

Related Functions

none

Example

To specify that the database memos for the report whose handle is *hRpt* were create with an Xbase memo editor:

```
setXbaseEditor (hRpt, 1);
```

writeRuntimeRecord

BOOL FAR PASCAL writeRuntimeRecord (int *hReport*, **LPSTR** *lpszControlFile***);**

hReport	Report handle.
lpszControlFile	Job-control-filename buffer.

Return Value

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

Description

Use **writeRuntimeRecord** to save all parameters for the report specified by *hReport* to the ASCII Runtime Control File specified by *lpszControlFile*. If *lpszControlFile* is the NULL pointer or contains the null string, **writeRuntimeRecord** will overwrite the Runtime 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 Runtime 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);
   }
}
```