

# Progress Report Builder Deployment Guide

© 2001 Progress Software Corporation. All rights reserved.

Progress® software products are copyrighted and all rights are reserved by Progress Software Corporation. This manual is also copyrighted and all rights are reserved. This manual may not, in whole or in part, be copied, photocopied, translated, or reduced to any electronic medium or machine-readable form without prior consent, in writing, from Progress Software Corporation.

The information in this manual is subject to change without notice, and Progress Software Corporation assumes no responsibility for any errors that may appear in this document.

The references in this manual to specific platforms supported are subject to change.

Progress, Progress Results, Provision and WebSpeed are registered trademarks of Progress Software Corporation in the United States and other countries. Apptivity, AppServer, ProVision Plus, SmartObjects, IntelliStream, and other Progress product names are trademarks of Progress Software Corporation.

SonicMQ is a trademark of Sonic Software Corporation in the United States and other countries.

Progress Software Corporation acknowledges the use of Raster Imaging Technology copyrighted by Snowbound Software 1993-1997 and the IBM XML Parser for Java Edition.

© IBM Corporation 1998-1999. All rights reserved. U.S. Government Users Restricted Rights — Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Progress is a registered trademark of Progress Software Corporation and is used by IBM Corporation in the mark Progress/400 under license. Progress/400 AND 400® are trademarks of IBM Corporation and are used by Progress Software Corporation under license.

Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Any other trademarks and/or service marks contained herein are the property of their respective owners.

.

May 2001




Product Code: 4519
Item Number: 81087W;9.1C

# Contents


**Preface** . . . . . **ix**
Purpose . . . . . ix
Audience . . . . . ix
Organization of This Manual . . . . . ix
Typographical Conventions . . . . . x
Syntax Notation . . . . . xi
Progress Messages . . . . . xiv
Other Useful Documentation . . . . . xvi
Getting Started . . . . . xvi
Development Tools . . . . . xvii
Reporting Tools . . . . . xviii
4GL . . . . . xix
Database . . . . . xix
DataServers . . . . . xx
SQL-89/Open Access . . . . . xx
SQL-92 . . . . . xxi
Deployment . . . . . xxi
WebSpeed . . . . . xxii
Reference . . . . . xxii

**1. Report Engine Basics** . . . . . **1–1**
1.1 How Report Engine Generates Reports . . . . . 1–2
1.2 How to Run Report Engine . . . . . 1–3
1.2.1 Table Interface . . . . . 1–3
1.2.2 PRINTRB Interface . . . . . 1–3
1.2.3 PRNTRB2 Interface . . . . . 1–4
1.2.4 Choosing Which Interface to Use . . . . . 1–4
1.3 Before You Get Started . . . . . 1–5

**2. Report Engine Administration . . . . . 2–1**
2.1 Generating a List of Reports . . . . . 2–2
2.2 Identifying the Default Printer . . . . . 2–4
2.3 Listing the Available Printers . . . . . 2–5
2.4 Accessing Initialization (.INI) Files . . . . . 2–7
2.5 Using Password Security . . . . . 2–9
2.6 Verifying Report Status . . . . . 2–10
2.6.1 Report Status File . . . . . 2–10
2.6.2 RB-STATUS Field . . . . . 2–13

**3. Report Engine Parameters . . . . . 3–1**
3.1 Parameter Basics . . . . . 3–2
3.2 Predefined Parameters . . . . . 3–2
3.2.1 A Brief Overview of the Predefined Parameters . . . . . 3–3
3.2.2 Complete Descriptions of Each Predefined Parameter . . . . . 3–4
3.3 User-defined Parameters . . . . . 3–20
3.3.1 Parameter Guidelines and Restrictions . . . . . 3–24
3.3.2 Prompting for User Input . . . . . 3–26
3.3.3 Incorporating Parameters in the Report Definition . . . . . 3–28

**4. Report Engine Table Interface . . . . . 4–1**
4.1 Table Interface Basics . . . . . 4–2
4.2 Table Interface Quick Start Example . . . . . 4–2
4.3 Creating the Report Engine Table . . . . . 4–4
4.3.1 Specifying Default Field Values . . . . . 4–6
4.4 Entering the Report Engine Parameter Values for the Reports . . . . . 4–6
4.5 Invoking Report Engine . . . . . 4–7
4.5.1 Command-line Syntax . . . . . 4–7
4.5.2 Command-line Parameters . . . . . 4–8
4.5.3 Methods of Invoking Report Engine . . . . . 4–15
4.6 Running Multiple Reports . . . . . 4–19
4.7 Running Reports from an Application . . . . . 4–19
4.7.1 Running a Report with a Simple Filter Override . . . . . 4–20
4.7.2 Prompting for User Input for the Printer Information . . . . . 4–22
4.7.3 Prompting for User Input for the Filter Information . . . . . 4–23
4.7.4 Generating a List of Reports, Printing the Reports, and Updating the RB-STATUS Field . . . . . 4–24
4.7.5 Generating a List of Reports, Prompting the User to Select Which Reports to Run, and Running the Reports . . . . . 4–26

**5. Report Engine PRINTRB and PRNTRB2 Interfaces . . . . . 5–1**
5.1 PRINTRB/PRNTRB2 Interface Basics . . . . . 5–2
5.2 PRINTRB Interface Quick Start Example . . . . . 5–2
5.3 PRNTRB2 Interface Example . . . . . 5–3
5.4 Using the PRINTRB/PRNTRB2 Parameters . . . . . 5–4
5.5 Running Report Engine from an Application . . . . . 5–6
5.5.1 Running a Report with a Simple Filter Override . . . . . 5–7
5.5.2 Prompting for User Input for the Printer Information . . . . . 5–8
5.5.3 Prompting for User Input for the Filter Information . . . . . 5–9
5.5.4 Running Multiple Reports in a Batch . . . . . 5–10
5.5.5 Running Multiple Reports and Prompting for User Input . . . . . 5–13
5.5.6 Overriding Database Connection Information . . . . . 5–15

**Index . . . . . Index–1**

**Figures**

Figure 1–1: How Report Engine Generates Reports . . . . . 1–2
Figure 2–1: Report Status File for Multiple Reports . . . . . 2–11

**Tables**


| | | |
|---|---|---|
| Table 1–1: | Table vs. PRINTRB/PRNTRB2 Interface Issues | 1–4 |
| Table 2–1: | Error Code Entries | 2–12 |
| Table 3–1: | Predefined Report Engine Parameters | 3–3 |
| Table 3–2: | RB-INCLUDE-RECORDS Values | 3–13 |
| Table 3–3: | RB-PRINT-DESTINATION Values | 3–16 |
| Table 3–4: | RB-STATUS Values | 3–18 |
| Table 3–5: | Report Engine Parameter Default Values | 3–25 |
| Table 4–1: | Required Field Names and Data Types | 4–4 |
| Table 4–2: | Optional Field Names and Data Types | 4–5 |
| Table 4–3: | Data Dictionary Default Initial Field Values | 4–6 |
| Table 4–4: | Valid Startup Parameter Locations | 4–8 |
| Table 4–5: | Report Engine Startup Parameters | 4–9 |
| Table 4–6: | RB-STATUS Values | 4–14 |
| Table 5–1: | Predefined Report Engine Parameters | 5–5 |

**Procedures**

reptlist.p . . . . . 2–3
defprint.p . . . . . 2–4
printlst.p . . . . . 2–6
progini.p . . . . . 2–8
password.p . . . . . 2–9
rbstart1.p . . . . . 4–3
rbfilt1.p . . . . . 4–21
rbprint1.p . . . . . 4–22
rbfilt2.p . . . . . 4–23
rbplist1.p . . . . . 4–25
rbplist2.p . . . . . 4–27
rbstart2.p . . . . . 5–3
rbstart3.p . . . . . 5–4
rbfilt3.p . . . . . 5–7
rbprint2.p . . . . . 5–8
rbfilt4.p . . . . . 5–9
rbbatch.p . . . . . 5–11
rbprompt.p . . . . . 5–13
rbdbconn.p . . . . . 5–15

# Preface

## Purpose

This guide explains how to generate and customize Report Builder reports using the Progress Report Engine. It explains the different interfaces to the Report Engine and how to access those interfaces from a 4GL program or MS-Windows icon. It also provides examples of how to incorporate reports into 4GL applications.

## Audience

This guide is for 4GL programmers and developers. Some knowledge of Progress is required, as well as a basic understanding of databases.

## Organization of This Manual

Chapter 1, "Report Engine Basics"

Describes how to use Report Engine to run Report Builder reports from Progress 4GL applications.

Chapter 2, "Report Engine Administration"

Describes Report Engine administration tasks, including generating a report list, identifying the default printer, listing the available printers, accessing initialization (.INI) files, using password security, and verifying report status.

Chapter 3, "Report Engine Parameters"

Explains how to use the Report Engine parameters to generate reports.

Chapter 4, "Report Engine Table Interface"

Describes how the Report Engine table interface stores the report parameters in a database table, and how you use that table to run multiple reports.

Chapter 5, "Report Engine PRINTRB and PRNTRB2 Interfaces"

Explains how to use the Report Engine PRINTRB and PRNTRB2 interfaces to invoke Report Engine from the Progress 4GL, using parameters instead of tables and fields.

## Typographical Conventions

This manual uses the following typographical conventions:

- **Bold typeface** indicates:
  - Commands or characters that the user types
  - That a word carries particular weight or emphasis
- *Italic typeface* indicates:
  - Progress variable information that the user supplies
  - New terms
  - Titles of complete publications
- `Monospaced typeface` indicates:
  - Code examples
  - System output
  - Operating system filenames and pathnames

The following typographical conventions are used to represent keystrokes:

- Small capitals are used for Progress key functions and generic keyboard keys.

  **END-ERROR**, **GET**, **GO**
  **ALT**, **CTRL**, **SPACEBAR**, **TAB**

- When you have to press a combination of keys, they are joined by a dash. You press and hold down the first key, then press the second key.

  **CTRL-X**

- When you have to press and release one key, then press another key, the key names are separated with a space.

  **ESCAPE H**
  **ESCAPE CURSOR-LEFT**

## Syntax Notation

The syntax for each component follows a set of conventions:

- Uppercase words are keywords. Although they are always shown in uppercase, you can use either uppercase or lowercase when using them in a procedure.

  In this example, ACCUM is a keyword:

  **SYNTAX**

  ```
  ACCUM aggregate expression
  ```

- Italics identify options or arguments that you must supply. These options can be defined as part of the syntax or in a separate syntax identified by the name in italics. In the ACCUM function above, the *aggregate* and *expression* options are defined with the syntax for the ACCUM function in the *Progress Language Reference*.

- You must end all statements (except for DO, FOR, FUNCTION, PROCEDURE, and REPEAT) with a period. DO, FOR, FUNCTION, PROCEDURE, and REPEAT statements can end with either a period or a colon, as in this example:

  ```
  FOR EACH Customer:
     DISPLAY Name.
  END.
  ```

- Square brackets ( **[ ]** ) around an item indicate that the item, or a choice of one of the enclosed items, is optional.

  In this example, STREAM *stream*, UNLESS-HIDDEN, and NO-ERROR are optional:

  **SYNTAX**

  ```
  DISPLAY [ STREAM stream ] [ UNLESS-HIDDEN ] [ NO-ERROR ]
  ```

  In some instances, square brackets are not a syntax notation, but part of the language.

  For example, this syntax for the INITIAL option uses brackets to bound an initial value list for an array variable definition. In these cases, normal text brackets ( [ ] ) are used:

  **SYNTAX**

  ```
  INITIAL [ constant [ , constant ] ... ]
  ```

  **NOTE:** The ellipsis ( **...** ) indicates repetition, as shown in a following description.

- Braces ( **{ }** ) around an item indicate that the item, or a choice of one of the enclosed items, is required.

  In this example, you must specify the items BY and *expression* and can optionally specify the item DESCENDING, in that order:

  **SYNTAX**

  ```
  { BY expression [ DESCENDING ] }
  ```

  In some cases, braces are not a syntax notation, but part of the language.

  For example, a called external procedure must use braces when referencing arguments passed by a calling procedure. In these cases, normal text braces ( { } ) are used:

  **SYNTAX**

  ```
  { &argument-name }
  ```

- A vertical bar ( **|** ) indicates a choice.

  In this example, EACH, FIRST, and LAST are optional, but you can only choose one:

  **SYNTAX**

  ```
  PRESELECT [ EACH | FIRST | LAST ] record-phrase
  ```

In this example, you must select one of *logical-name* or *alias*:

**SYNTAX**

```
CONNECTED ( { logical-name | alias } )
```

- Ellipses ( **. . .** ) indicate that you can choose one or more of the preceding items. If a group of items is enclosed in braces and followed by ellipses, you must choose one or more of those items. If a group of items is enclosed in brackets and followed by ellipses, you can optionally choose one or more of those items.

  In this example, you must include two expressions, but you can optionally include more. Note that each subsequent expression must be preceded by a comma:

  **SYNTAX**

  ```
  MAXIMUM ( expression , expression [ , expression ] ... )
  ```

  In this example, you must specify MESSAGE, then at least one of *expression* or SKIP, but any additional number of *expression* or SKIP is allowed:

  **SYNTAX**

  ```
  MESSAGE { expression | SKIP [ (n) ] } ...
  ```

  In this example, you must specify {*include-file*, then optionally any number of *argument* or &*argument-name* = "*argument-value*", and then terminate with }:

  **SYNTAX**

  ```
  { include-file
      [ argument | &argument-name = "argument-value" ] ... }
  ```

- In some examples, the syntax is too long to place in one horizontal row. In such cases, **optional** items appear individually bracketed in multiple rows in order, left-to-right and top-to-bottom. This order generally applies, unless otherwise specified. **Required** items also appear on multiple rows in the required order, left-to-right and top-to-bottom. In cases where grouping and order might otherwise be ambiguous, braced (required) or bracketed (optional) groups clarify the groupings.

  In this example, WITH is followed by several optional items:

  **SYNTAX**

```
WITH [ ACCUM max-length ] [ expression DOWN ]
  [ CENTERED ] [ n COLUMNS ] [ SIDE-LABELS ]
  [ STREAM-IO ]
```

  In this example, ASSIGN requires one of two choices: either one or more of *field*, or one of *record*. Other options available with either *field* or *record* are grouped with braces and brackets. The open and close braces indicate the required order of options:

  **SYNTAX**

```
ASSIGN {   { [ FRAME frame ]
               { field [ = expression ] }
               [ WHEN expression ]
           } ...
         | { record [ EXCEPT field ... ] }
       }
```

## Progress Messages

Progress displays several types of messages to inform you of routine and unusual occurrences:

- Execution messages inform you of errors encountered while Progress is running a procedure (for example, if Progress cannot find a record with a specified index field value).
- Compile messages inform you of errors found while Progress is reading and analyzing a procedure prior to running it (for example, if a procedure references a table name that is not defined in the database).

- Startup messages inform you of unusual conditions detected while Progress is getting ready to execute (for example, if you entered an invalid startup parameter).

After displaying a message, Progress proceeds in one of several ways:

- Continues execution, subject to the error-processing actions that you specify, or that are assumed, as part of the procedure. This is the most common action taken following execution messages.
- Returns to the Progress Procedure Editor so that you can correct an error in a procedure. This is the usual action taken following compiler messages.
- Halts processing of a procedure and returns immediately to the Procedure Editor. This does not happen often.
- Terminates the current session.

Progress messages end with a message number in parentheses. In this example, the message number is 200:

```
** Unknown table name table. (200)
```

Use Progress online help to get more information about Progress messages. Many Progress tools include the following Help menu options to provide information about messages:

- Choose Help→ Recent Messages to display detailed descriptions of the most recent Progress message and all other messages returned in the current session.
- Choose Help→ Messages, then enter the message number to display a description of any Progress message. (If you encounter an error that terminates Progress, make a note of the message number before restarting.)
- In the Procedure Editor, press the **HELP** key (**F2** or **CTRL-W**).

## Other Useful Documentation

This section lists Progress Software Corporation documentation that you might find useful. Unless otherwise specified, these manuals support both Windows **and** Character platforms and are provided in electronic documentation format on CD-ROM.

### Getting Started

*Progress Electronic Documentation Installation and Configuration Guide* (Hard copy only)

A booklet that describes how to install the Progress EDOC viewer and collection on UNIX and Windows.

*Progress Installation and Configuration Guide Version 9 for UNIX*

A manual that describes how to install and set up Progress Version 9.1 for the UNIX operating system.

*Progress Installation and Configuration Guide Version 9 for Windows*

A manual that describes how to install and set up Progress Version 9.1 for all supported Windows and Citrix MetaFrame operating systems.

*Progress Version 9 Product Update Bulletin*

A guide that provides a brief description of each new feature of the release. The booklet also explains where to find more detailed information in the documentation set about each new feature.

*Progress Application Development Environment — Getting Started* (Windows only)

A practical guide to graphical application development within the Progress Application Development Environment (ADE). This guide includes an overview of the ADE and its tools, an overview of Progress SmartObject technology, and tutorials and exercises that help you better understand SmartObject technology and how to use the ADE to develop applications.

*Progress Language Tutorial for Windows* and *Progress Language Tutorial for Character*

Platform-specific tutorials designed for new Progress users. The tutorials use a step-by-step approach to explore the Progress application development environment using the 4GL.

*Progress Master Glossary for Windows* and *Progress Master Glossary for Character* (EDOC only)

Platform-specific master glossaries for the Progress documentation set. These books are in electronic format only.

*Progress Master Index and Glossary for Windows* and *Progress Master Index and Glossary for Character* (Hard copy only)

Platform-specific master indexes and glossaries for the Progress hard-copy documentation set.

*Progress Startup Command and Parameter Reference*

A reference manual that describes the Progress startup commands and parameters in alphabetical order.

*Welcome to Progress* (Hard copy only)

A booklet that explains how Progress software and media are packaged. An icon-based map groups the documentation by functionality, providing an overall view of the documentation set. *Welcome to Progress* also provides descriptions of the various services Progress Software Corporation offers.

### Development Tools

*Progress ADM 2 Guide*

A guide to using the Application Development Model, Version 2 (ADM 2) application architecture to develop Progress applications. It includes instructions for building and using Progress SmartObjects.

*Progress ADM 2 Reference*

A reference for the Application Development Model, Version 2 (ADM 2) application. It includes descriptions of ADM 2 functions and procedures.

*Progress AppBuilder Developer's Guide* (Windows only)

A programmer's guide to using the Progress AppBuilder visual layout editor. AppBuilder is a Rapid Application Development (RAD) tool that can significantly reduce the time and effort required to create Progress applications.

*Progress Basic Database Tools* (Character only; information for Windows is in online help)

A guide for the Progress Database Administration tools, such as the Data Dictionary.

*Progress Basic Development Tools* (Character only; information for Windows is in online help)

A guide for the Progress development toolset, including the Progress Procedure Editor and the Application Compiler.

*Progress Debugger Guide*

A guide for the Progress Application Debugger. The Debugger helps you trace and correct programming errors by allowing you to monitor and modify procedure execution as it happens.

*Progress Help Development Guide* (Windows only)

A guide that describes how to develop and integrate an online help system for a Progress application.

*Progress Translation Manager Guide* (Windows only)

A guide that describes how to use the Progress Translation Manager tool to manage the entire process of translating the text phrases in Progress applications.

*Progress Visual Translator Guide* (Windows only)

A guide that describes how to use the Progress Visual Translator tool to translate text phrases from procedures into one or more spoken languages.

### Reporting Tools

*Progress Report Builder Tutorial* (Windows only)

A tutorial that provides step-by-step instructions for creating eight sample Report Builder reports.

*Progress Report Builder User's Guide* (Windows only)

A guide for generating reports with the Progress Report Builder.

*Progress Results Administration and Development Guide* (Windows only)

A guide for system administrators that describes how to set up and maintain the Results product in a graphical environment. This guide also describes how to program, customize, and package Results with your own products. In addition, it describes how to convert character-based Results applications to graphical Results applications.

*Progress Results User's Guide for Windows* and *Progress Results User's Guide for UNIX*

Platform-specific guides for users with little or no programming experience that explain how to query, report, and update information with Results. Each guide also helps advanced users and application developers customize and integrate Results into their own applications.

### 4GL

*Building Distributed Applications Using the Progress AppServer*

A guide that provides comprehensive information about building and implementing distributed applications using the Progress AppServer. Topics include basic product information and terminology, design options and issues, setup and maintenance considerations, 4GL programming details, and remote debugging.

*Progress External Program Interfaces*

A guide to accessing non-Progress applications from Progress. This guide describes how to use system clipboards, UNIX named pipes, Windows dynamic link libraries, Windows dynamic data exchange, Windows ActiveX controls, and the Progress Host Language Call Interface to communicate with non-Progress applications and extend Progress functionality.

*Progress Internationalization Guide*

A guide to developing Progress applications for markets worldwide. The guide covers both internationalization—writing an application so that it adapts readily to different locales (languages, cultures, or regions)—and localization—adapting an application to different locales.

*Progress Language Reference*

A three-volume reference set that contains extensive descriptions and examples for each statement, phrase, function, operator, widget, attribute, method, and event in the Progress language.

*Progress Programming Handbook*

A two-volume handbook that details advanced Progress programming techniques.

### Database

*Progress Database Design Guide*

A guide that uses a sample database and the Progress Data Dictionary to illustrate the fundamental principles of relational database design. Topics include relationships, normalization, indexing, and database triggers.

*Progress Database Administration Guide and Reference*

This guide describes Progress database administration concepts and procedures. The procedures allow you to create and maintain your Progress databases and manage their performance.

### DataServers

Progress DataServer Guides

These guides describe how to use the DataServers to access non-Progress databases. They provide instructions for building the DataServer modules, a discussion of programming considerations, and a tutorial. Each DataServer has its own guide, for example, the *Progress DataServer for ODBC Guide*, the *Progress DataServer for ORACLE Guide*, or the *Progress/400 Product Guide*.

MERANT ODBC Branded Driver Reference

The Enterprise DataServer for ODBC includes MERANT ODBC drivers for all the supported data sources. For configuration information, see the MERANT documentation, which is available as a PDF file in *installation-path*\odbc. To read this file you must have the Adobe Acrobat Reader Version 3.1 or higher installed on your system. If you do not have the Adobe Acrobat Reader, you can download it from the Adobe Web site at: http://www.adobe.com/prodindex/acrobat/readstep.html.

### SQL-89/Open Access

*Progress Embedded SQL-89 Guide and Reference*

A guide to Progress Embedded SQL-89 for C, including step-by-step instructions on building ESQL-89 applications and reference information on all Embedded SQL-89 Preprocessor statements and supporting function calls. This guide also describes the relationship between ESQL-89 and the ANSI standards upon which it is based.

*Progress Open Client Developer's Guide*

A guide that describes how to write and deploy Java and ActiveX applications that run as clients of the Progress AppServer. The guide includes information about how to expose the AppServer as a set of Java classes or as an ActiveX server.

*Progress SQL-89 Guide and Reference*

A user guide and reference for programmers who use interactive Progress/SQL-89. It includes information on all supported SQL-89 statements, SQL-89 Data Manipulation Language components, SQL-89 Data Definition Language components, and supported Progress functions.

### SQL-92

*Progress Embedded SQL-92 Guide and Reference*

A guide to Progress Embedded SQL-92 for C, including step-by-step instructions for building ESQL-92 applications and reference information about all Embedded SQL-92 Preprocessor statements and supporting function calls. This guide also describes the relationship between ESQL-92 and the ANSI standards upon which it is based.

*Progress JDBC Driver Guide*

A guide to the Java Database Connectivity (JDBC) interface and the Progress SQL-92 JDBC driver. It describes how to set up and use the driver and details the driver's support for the JDBC interface.

*Progress ODBC Driver Guide*

A guide to the ODBC interface and the Progress SQL-92 ODBC driver. It describes how to set up and use the driver and details the driver's support for the ODBC interface.

*Progress SQL-92 Guide and Reference*

A user guide and reference for programmers who use Progress SQL-92. It includes information on all supported SQL-92 statements, SQL-92 Data Manipulation Language components, SQL-92 Data Definition Language components, and Progress functions. The guide describes how to use the Progress SQL-92 Java classes and how to create and use Java stored procedures and triggers.

### Deployment

*Progress Client Deployment Guide*

A guide that describes the client deployment process and application administration concepts and procedures.

*Progress Developer's Toolkit*

A guide to using the Developer's Toolkit. This guide describes the advantages and disadvantages of different strategies for deploying Progress applications and explains how you can use the Toolkit to deploy applications with your selected strategy.

*Progress Portability Guide*

A guide that explains how to use the Progress toolset to build applications that are portable across all supported operating systems, user interfaces, and databases, following the Progress programming model.

### WebSpeed

*Getting Started with WebSpeed*

Provides an introduction to the WebSpeed Workshop tools for creating Web applications. It introduces you to all the components of the WebSpeed Workshop and takes you through the process of creating your own Intranet application.

*WebSpeed Installation and Configuration Guide*

Provides instructions for installing WebSpeed on Windows and UNIX systems. It also discusses designing WebSpeed environments, configuring WebSpeed Brokers, WebSpeed Agents, and the NameServer, and connecting to a variety of data sources.

*WebSpeed Developer's Guide*

Provides a complete overview of WebSpeed and the guidance necessary to develop and deploy WebSpeed applications on the Web.

*WebSpeed Product Update Bulletin*

A booklet that provides a brief description of each new feature of the release. The booklet also explains where to find more detailed information in the documentation set about each new feature.

*Welcome to WebSpeed* (Hard copy only)

A booklet that explains how WebSpeed software and media are packaged. *Welcome to WebSpeed* also provides descriptions of the various services Progress Software Corporation offers.

### Reference

*Pocket Progress* (Hard copy only)

A reference that lets you quickly look up information about the Progress language or programming environment.

*Pocket WebSpeed* (Hard copy only)

A reference that lets you quickly look up information about the SpeedScript language or the WebSpeed programming environment.

# 1

# Report Engine Basics

Report Engine allows you to run reports created by Report Builder from Progress 4GL applications. This chapter describes:

- How Report Engine generates reports
- How to run Report Engine
- What to do before you get started

## 1.1 How Report Engine Generates Reports

Report Engine allows you to run reports from within an application. However, before Report Engine can produce a report, you must first create the report definition with Report Builder. The *report definition* defines the report characteristics that tell Report Builder and Report Engine where to find the data and how to display and print the data. See the *Progress Report Builder User's Guide* for a complete explanation of how to create a report definition with Report Builder.

Report Engine uses the specifications from the report definition, which includes the database connection information, to generate reports. You can specify overrides to the report specifications when you generate reports, using the Report Engine parameters. *Report Engine parameters* are parameters that allow you to provide and override specific portions of report information. See Chapter 3, "Report Engine Parameters," for a complete description of how to use the Report Engine parameters.

Figure 1–1 illustrates how Report Engine generates reports using the database information, report definition, and Report Engine parameters.

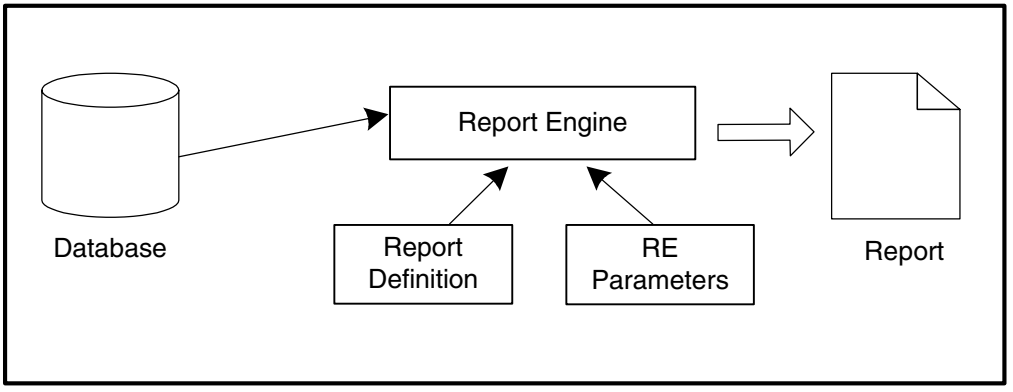



**Figure 1–1: How Report Engine Generates Reports**

Although you can generate reports with either Report Builder or Report Engine, there are two important reasons you might use Report Engine instead of Report Builder:

- To create a report with Report Builder, you must be running Report Builder. However, you can run previously defined reports with Report Engine without access to Report Builder. This allows users who do not have access to Report Builder to run reports.
- You can generate reports by invoking Report Engine from a 4GL application. This means that you can write applications to run batches of reports, run reports at specific times, or allow users to change reports and specify report overrides using an interface that you can design within an application.

You can override specifications in the report definition with Report Engine without changing the report definition. This means, for example, that you can create a report definition to display data from a test database and then override the database specifications with the connection information for a production database.

## 1.2 How to Run Report Engine

There are three Report Engine interfaces you can use:

- Table interface
- PRINTRB interface
- PRNTRB2 interface

The following sections provide a brief overview of the three Report Engine interfaces. See Chapter 4, "Report Engine Table Interface," and Chapter 5, "Report Engine PRINTRB and PRNTRB2 Interfaces," for complete descriptions of each interface.

### 1.2.1 Table Interface

To use the table interface, you create a database record that specifies values for all the report parameters, and then Report Engine reads the record to obtain the parameter values. You can also create more than one database record—one for each report— and Report Engine will run all the reports. When you use the Report Engine table interface, you can run it either from a 4GL application or from an MS-Windows icon using the standard Progress client command line parameters.

You must use the table interface for full access to Version 8 or greater Report Builder functionality.

### 1.2.2 PRINTRB Interface

The PRINTRB interface allows you to specify nineteen report parameters in an application. You can run it only from a 4GL application, which means that you must be in a Progress session to run the Report Engine PRINTRB interface. The PRINTRB interface does not support some of the Version 8 or greater Report Builder functionality.

### 1.2.3 PRNTRB2 Interface

The PRNTRB2 interface allows you to specify twenty report parameters in an application. You can run it only from a 4GL application, which means that you must be in a Progress session to run the Report Engine PRNTRB2 interface. PRNTRB2 allows you to run multiple instances of Report Engine simultaneously in the same directory. The PRNTRB2 interface does not support some of the Version 8 or greater Report Builder functionality.

### 1.2.4 Choosing Which Interface to Use

Table 1–1 lists the issues you must consider when deciding which interface to use.

**Table 1–1: ComparingTable and PRINTRB/PRNTRB2 Interface Issues**

| Issue | Table Interface | PRINTRB Interface | PRNTRB2 Interface |
|---|---|---|---|
| Can run from an application. | √ | √ | √ |
| Accepts command line parameters from startup parameter files (*startup.pf* and *rbstart.pf*). | √ | √ | √ |
| Can run from an MS-Windows icon and accepts command line parameters directly from the command line (including database connections); this means that it can run from outside of Progress. | √ | – | – |
| Can run multiple reports with single invocation of the Report Engine; this means that you do not have to start up and shut down Report Engine for each report. | √ | – | – |
| Supports multiple database connections. | √ | – | – |
| Can run multiple instances of Report Engine simultaneously | √ | – | √ |

## 1.3 Before You Get Started

Before you start Progress Report Engine, review the following checklist:

- To have access to all the fonts and formatting specified in the report definitions that you intend to use, verify that you have an MS-Windows printer driver installed. (The physical printer does not have to be installed if you are only previewing reports on the screen or printing them to files). If you do not have a printer driver installed, Report Engine will not have access to the font information that it requires to generate reports.
- Report Engine must be installed on the machine where you are running it.
- The report libraries containing the reports you have designed using Report Builder must be on the machine where you are running Report Engine.
- The databases you designed your reports for must be accessible by Report Engine.
- If your reports use any user-defined functions, the `PRORB.UDF` file in which the UDF definitions are stored must be on the machine where you are running Report Engine.
- Any memo files attached to your reports must be on the machine where you are running Report Engine.

If you are using the table interface, you must also:

- Create a table in which to place the report parameters, in a new or existing database. This table is the *Report Engine table* and the database that contains it is the *Report Engine database*. The table must have fields for each of the eighteen required parameters. See Chapter 4, "Report Engine Table Interface," for more information about creating this database. Progress supplies a data definition file that contains a sample table. The Report Engine record must be created in an actual database; it cannot be created in a temporary table or a work file.
- Connect the application running Report Engine to the above database.
- Create one or more records containing values for the report parameters.

Once you have performed these tasks, you can invoke Report Engine from the 4GL or create an icon that invokes Report Engine using the Report Engine table.

# 2

# Report Engine Administration

This chapter describes Report Engine administration tasks, including:

- Generating a report list
- Identifying the default printer
- Listing the available printers
- Accessing initialization (.INI) files
- Using password security
- Verifying report status

## 2.1 Generating a List of Reports

You can generate a list of reports in a report library using the aderb/_getname.p procedure. This procedure also calculates the number of reports in the library.

Follow these steps to generate a list of reports in a report library:

1 ♦ Create the report definitions in Report Builder.

2 ♦ Use the following command in the Progress 4GL application:

```
RUN aderb/_getname.p (library-pathname, OUTPUT report-list,
     OUTPUT report-count)
```

The following list describes the arguments for the _getname.p procedure:

*library-pathname*

The full path of the report library you want to read.

*report-list*

The character variable that will display the report names as a comma separated list.

*report-count*

The integer variable that will receive the number of reports in the library.

For example, to list all the reports in the acctslib library and receive the number of reports in this library, run the reptlist.p procedure:

**reptlist.p**

```
/* 1 */
  DEF VAR report-list AS CHARACTER.
  DEF VAR report-count AS INTEGER.
  DEF VAR report AS CHAR VIEW-AS SELECTION-LIST INNER-LINES 10
    INNER-CHARS 32.

/* 2 */
  FORM "There were" report-count "Reports; please select one: "  SKIP
    report
    WITH FRAME x NO-LABELS.

/* 3 */
  RUN aderb/_getname.p ("c:\acctslib.prl", OUTPUT report-list,
    OUTPUT report-count).

    report = ?.
    report:list-items = report-list.
    DISPLAY report-count WITH FRAME x.
    UPDATE report WITH FRAME x.
```

The commented numbers correspond to the following step-by-step descriptions:

1. Define the variables that will display the report names and the number of reports.
2. Format a selection list to display the list of report names.
3. Call the _getname.p procedure and specify first the report library, second, the variable in which to return the report names, and lastly, the variable in which to return the number of reports in the library.

## 2.2 Identifying the Default Printer

You can identify the current default printer and port for MS-Windows from the Progress 4GL using the aderb/_prdef.p procedure. This procedure is useful for assigning printers at run time.

Use the following command in the Progress 4GL application:

```
RUN aderb/_prdef.p (OUTPUT printer-name, OUTPUT port-name,
      OUTPUT success)
```

The following list describes the arguments for the _prdef.p procedure:

*printer-name*

The character variable that will receive the name of the printer.

*port-name*

The character variable that will receive the name of the port.

*success*

The logical variable that gets assigned a TRUE value (yes) if Progress returns a string of zero or more characters and a FALSE value if no string is returned.

For example, run the defprint.p procedure:

**defprint.p**

```
/* 1 */
  DEF VAR printer AS CHARACTER FORMAT "x(45)".
  DEF VAR port AS CHARACTER FORMAT "x(20)".
  DEF VAR success AS LOGICAL.

/* 2 */
  RUN aderb/_prdef.p (OUTPUT printer, OUTPUT port, OUTPUT success).

/* 3 */
  IF success THEN DO:
    DISPLAY printer port.
    PAUSE.
  END
  ELSE MESSAGE "_prdef failed" VIEW-AS ALERT-BOX ERROR.
```

The commented numbers correspond to the following step-by-step descriptions:

1. Define the variables that will display the printer name and port name. Also define a logical variable that will test whether the printer information is available. If the procedure returns a string, `_prdef.p` sets the variable to True; if no string is returned, `_prdef.p` sets the variable to False.

2. Call the `_prdef.p` procedure and specify the printer, port, and logical variables.

3. Test the value in the *success* variable, then display the appropriate information.

## 2.3 Listing the Available Printers

You can generate a list of available printers and ports from the Progress 4GL by using the `aderb/_prlist.p` procedure.

Use the following command in the Progress 4GL application:

```
RUN aderb/_prlist.p (OUTPUT printer-list, OUTPUT port-list,
      OUTPUT printer-count)
```

The following list describes the arguments for the `_prlist.p` procedure:

*printer-list*

The character variable that will receive the printer names as a comma-separated list.

*port-list*

The character variable that will receive the port names as a comma-separated list.

*printer-count*

The integer variable that counts the number of printers.

For example, run the printlst.p procedure:

**printlst.p**

```
/* 1 */
  DEF VAR printer-list AS CHARACTER NO-UNDO.
  DEF VAR port-list AS CHARACTER NO-UNDO.
  DEF VAR i AS INTEGER NO-UNDO.
  DEF VAR printer-count AS INTEGER NO-UNDO.
  DEF VAR printer AS CHARACTER NO-UNDO FORMAT "x(32)".
  DEF VAR port AS CHARACTER NO-UNDO FORMAT "x(20)".

/* 2 */
  RUN aderb/_prlist.p (OUTPUT printer-list, OUTPUT port-list,
    OUTPUT printer-count).

/* 3 */
  REPEAT i = 1 TO printer-count:
    printer = entry (i, printer-list).
    port = entry (i, port-list).
  DISPLAY i ")" printer port.
  END.

  PAUSE.
```

The commented numbers correspond to the following step-by-step descriptions:

1. Define the variables that will display the printer and port names and the number of printers.

2. Call the _prlist.p procedure and specify the printer, port, and printer count variables.

3. Display each of the printers and ports.

## 2.4 Accessing Initialization (.INI) Files

To access initialization file information from the Progress 4GL, you can use the _getini.p procedure. Because the status file created by Report Engine has the same format as an initialization file, you can use the _getini.p procedure to read the information out of the status file. Use the following command in the Progress 4GL application:

```
RUN aderb/_getini.p (section-name, entry-name, entry-default,
       OUTPUT entry-value, ini-pathname, OUTPUT success)
```

The following list describes the arguments for the _getini.p procedure:

*section-name*

The name of the section in the initialization file. In the Report Engine status file, this is also the name of the report being run.

*entry-name*

The name of the entry that you want. For example, RO-ERROR-CODE or RO-ERROR-MESSAGE in the report status file.

*entry-default*

The default value to be used if the entry is not found in the specified section.

*entry-value*

The character variable that will display the value of the specified entry.

*ini-pathname*

The full pathname to the initialization file being used. For example, with the Report Engine table interface, this is the file specified by the Report Status File (-rbstatfile) parameter.

*success*

The logical variable that gets assigned a TRUE value (yes) if Progress returns a string of zero of more characters and a FALSE value if no string is returned. Currently, this parameter always gets assigned a TRUE value (yes). The parameter is included to allow for future enhancements.

The progini.p procedure is an example of how to run the _getini.p procedure:

**progini.p**

```
/* 1 */
  DEF VAR result AS CHARACTER FORMAT "X(70)".
  DEF VAR success AS LOGICAL.

/* 2 */
  RUN aderb/_getini.p ("Customer Report", "RO-ERROR-CODE", "",
  OUTPUT result,"c:\examples\rbrun.out", OUTPUT success).

/* 3 */
  IF success THEN DO:
    DISPLAY result.
    PAUSE.
  END
/* 4 */
  ELSE MESSAGE "It failed" VIEW-AS ALERT-BOX ERROR.
```

The commented numbers correspond to the following step-by-step descriptions:

1. Define the variables that will pass the character value and logical value for the _getini.p procedure.

2. Call aderb/_getini.p and specify all six arguments, including the variables to pass the result and return values.

3. Test the value in the *success* variable, then display the appropriate information.

4. Currently, the ELSE code will never be executed. The test shown will never fail—*success* always has the value *true*. This is an artifact of how the underlying MS-Windows function behaves.

## 2.5 Using Password Security

To add security for user passwords with the table interface, use the aderb/_rbpwenc.p procedure to encode the passwords, then specify the Encoded Password (-rbP) parameter in the database connection string either on the command line or in the RB-DB-CONNECTION field of the Report Engine table. When you use the -rbP parameter, Report Engine accepts encoded passwords and deciphers the encoding. You can specify -rbP either on the command line or in place of the Password (-P) parameter in the RB-DB-CONNECTION field of the Report Engine table.

Progress Software Corporation supplies the compiled code for the _rbpwenc procedure. The procedure accepts two character arguments:

- An input argument, which is the password
- An output argument, which is the the encoded password

The password.p. procedure shows how to use the aderb/_rbpwenc procedure:

**password.p**

```
DEF VAR encodepw AS CHARACTER.

DO TRANSACTION:

  CREATE RBREPORT.

  ASSIGN
    RBREPORT.RB-REPORT-LIBRARY = "rbsample.prl"
    RBREPORT.RB-REPORT-NAME = "Customer Discount"
    RBREPORT.RB-PRINT-DESTINATION = "D".
    RELEASE RBREPORT.
END.

RUN aderb/_rbpwenc.p ("password", OUTPUT encodepw).

RUN aderb/_prore.p (FALSE, "-db sample.db -S demosv1
  -H localhost -N tcp -rbdel -U username -rbP " + encodepw ).
```

**CAUTION:** Although _rbpwenc encodes users' passwords making it more difficult for a user to obtain the password required to connect to a database, other users can still copy the encoded password and use it to generate their own reports. Progress Software provides _rbpwenc to make your password more secure, but **it does not guarantee that** _rbpwenc **is unbreakable**.

## 2.6 Verifying Report Status

There are two ways that Report Engine reports status information:

- The report status file
- The RB-STATUS field (table interface only)

Report Engine writes status and error messages to one or both of these places depending on which interface and command-line parameters you use.

### 2.6.1 Report Status File

The first location where Report Engine outputs status information is a text file called the *report status file*. The file's name and location depend upon which interface you use.

If you use the PRINTRB interface, Report Engine automatically writes report status information to the report status file, called `RBRUN.OUT` in the current directory.

If you use the PRNTRB2 interface, you must set the RB-STATUS-FILE parameter. The report status file is created and has the same format as the `RBRUN.OUT` file. If you do not indicate the filename, this file is not generated. The file is written to either the directory specified in the RB-STATUS-FILE or the current directory if none is specified. See Chapter 5, "Report Engine PRINTRB and PRNTRB2 Interfaces," for a complete description of the Report Status File parameter.

However, if you use the table interface, you can choose whether to have Report Engine create a report status file, using the Report Status File (-rbstatfile) parameter. This parameter specifies the file name and directory of the report status file. See Chapter 4, "Report Engine Table Interface," for a complete description of the Report Status File parameter.

After running Report Engine, you can check the status file for information about Report Engine processing. If Report Engine encountered an error, the file contains an error message that explains why a report was terminated, as well as an error code that lets you determine the type of error. The status file also contains the number of pages printed, so a report can be restarted where it left off.

If you set the RB-DISPLAY-ERRORS parameter to true, Report Engine also displays error messages on the screen as it executes. For a complete description of the RB-DISPLAY-ERRORS parameter, see Chapter 3, "Report Engine Parameters,"

To avoid confusing old and new status files, delete any existing status files before starting Report Engine.

The report status file is in Windows initialization (.INI) style format and has the header [*report-name*]. If Report Engine is running multiple reports, the report status file contains information about all the reports that were run (table interface only). Each report's output appears in the same format as a separate .INI file section. Figure 2–1 shows a report status file for multiple reports.

```
[Customer Report]
RO-ERROR-CODE=N
RO-ERROR-MESSAGE=
RO-PAGES=1


[Order Report]
RO-ERROR-CODE=J
RO-ERROR-MESSAGE=Invalid report library
Ro-PAGES=0
```

**Figure 2–1: Report Status File for Multiple Reports**

There are three entries in Report Engine status file:

- RO-ERROR-CODE
- RO-ERROR-MESSAGE
- RO-PAGES

Entries appear one per line in the following format:

```
entry = value
```

The following sections describe each of the three entries.

### RO-ERROR-CODE

Table 2–1 describes the possible error code entries.

**Table 2–1: Error Code Entries**

| Entry | Meaning |
|---|---|
| N | Report Engine completed without error. The RO-ERROR-MESSAGE value is blank. |
| C | The user selected Cancel to cancel a report. The RO-ERROR-MESSAGE message is “Report cancelled.” |
| J | There is an error in the Report Engine command or in the Report Engine record. See the message in the RO-ERROR-MESSAGE entry. |
| R | An error occurred while attempting to process the report. See the message in the RO-ERROR-MESSAGE entry. |

### RO-ERROR-MESSAGE

The error message value depends upon the type of error. The following table lists what type of message appears for different errors:

| If the error message is . . . | Then . . . |
|---|---|
| Blank | Report Engine completed without error. |
| Report cancelled | The user selected Cancel to cancel a report. |
| Other | An error occurred while trying to process the report. This might be caused by invalid information in the report definition, such as an incorrect database name, an error in the Report Engine parameters (for example, a misspelled field name in a filter expression), or an external factor (for example, a server or printer is unavailable). |

### RO-PAGES

This entry contains the number of the last page completed in the report. You can use this number to restart a terminated report at the page where the error occurred.

For example, if you are printing pages 10 through 20 of a report and the printer jams on page 15, this entry contains 14 (the number of the last page that printed successfully). If RO-PAGES contains 14, you can restart the report at page 15 by entering 15 in the RB-BEGIN-PAGE parameter and 20 in the RB-END-PAGE parameter.

## 2.6.2 RB-STATUS Field

The second location where Report Engine outputs status information is the RB-STATUS field in the Report Engine table. However, Report Engine writes report status and error messages to the RB-STATUS field only if you specify the Report Update Status (-rbupds) parameter. See Chapter 4, “Report Engine Table Interface,” for more information about the RB-STATUS field.

# 3

# Report Engine Parameters

This chapter describes how to use the Report Engine parameters to generate reports. Specifically, it describes:

- Parameter basics
- Predefined parameters
- User-defined parameters

## 3.1 Parameter Basics

When you run a report with Report Engine, you can provide or override many of the report characteristics using report parameters. These parameters identify the report library that contains the report to be run and the report name, and optionally override some of the report characteristics stored in the report definition. There is also an output parameter that can be used with the table interface to provide report status information.

There are two types of parameters:

- Predefined
- User-defined

The following sections describe how to use both types of parameters. Because Report Engine has three different interfaces, take note of the interface-specific information. See Chapter 4, "Report Engine Table Interface," and Chapter 5, "Report Engine PRINTRB and PRNTRB2 Interfaces," for information about the different interfaces.

## 3.2 Predefined Parameters

*Predefined parameters* are parameters defined by Report Engine to identify the report you want to run and to control frequently changed report features, such as filters and print information, allowing you to customize your reports. There is also an output parameter that provides status information.

Report Engine provides predefined parameters to control report processing. Depending on the interface you use, you specify parameters as arguments to a procedure or as values stored in a database table. See Chapter 4, "Report Engine Table Interface," or Chapter 5, "Report Engine PRINTRB and PRNTRB2 Interfaces," for more information about using the parameters with each interface.

### 3.2.1 A Brief Overview of the Predefined Parameters

Table 3–1 lists the predefined Report Engine parameters. Each parameter name has the prefix RB-, indicating that it is a Report Builder parameter. In the Data Type column, the letter **C** represents the CHARACTER data type, **I** represents INTEGER, and **L** represents LOGICAL. Parameters that require character values can contain uppercase, lowercase, or mixed-case letters.

**Table 3–1: Predefined Report Engine Parameters** *(1 of 2)*

| Parameter Name | Contents | Data Type | Interface |
|---|---|---|---|
| RB-REPORT-LIBRARY | Report library name | C | All |
| RB-REPORT-NAME | Report name | C | All |
| RB-DB-CONNECTION | Database connection override string | C | All |
| RB-INCLUDE-RECORDS | Filter flag | C | All |
| RB-FILTER | Filter expression | C | All |
| RB-MEMO-FILE | Memo filename | C | All |
| RB-PRINT-DESTINATION | Print destination | C | All |
| RB-PRINTER-NAME | Printer name | C | All |
| RB-PRINTER-PORT | Printer port | C | All |
| RB-OUTPUT-FILE | Output filename | C | All |
| RB-NUMBER-COPIES | Number of copies | I | All |
| RB-BEGIN-PAGE | Beginning page number | I | All |
| RB-END-PAGE | Ending page number | I | All |
| RB-TEST-PATTERN | Test pattern flag | L | All |
| RB-WINDOW-TITLE | Window title | C | All |
| RB-DISPLAY-ERRORS | Display error flag | L | All |

**Table 3–1: Predefined Report Engine Parameters** *(2 of 2)*

| Parameter Name | Contents | Data Type | Interface |
|---|---|---|---|
| RB-DISPLAY-STATUS | Display status flag | L | All |
| RB-NO-WAIT | Synchronous or asynchronous processing | L | PRINTRB<br>PRNTRB2 |
| RB-OTHER-PARAMETERS | User-defined parameters | C | All |
| RB-TAG | Which reports to generate | C | Table |
| RB-STATUS | Status of a report | C | Table |
| RB-STATUS-FILE | Report status filename | C | PRNTRB2 |

For the PRINTRB and PRNTRB2 interfaces, you must enter the parameter values in the order in which the parameters appear in Table 3–1. However, for the table interface, the parameters are represented by fields in a table and can be assigned values in any order. Regardless of which interface you use, you must specify values for the RB-REPORT-LIBRARY and RB-REPORT-NAME parameters.

### 3.2.2 Complete Descriptions of Each Predefined Parameter

This section describes each of the predefined Report Engine parameters in alphabetical order.

`RB-BEGIN-PAGE`

Specifies the starting page number for the report. The value you specify for RB-BEGIN-PAGE must be less than or equal to the report's ending page, which can be saved in the report definition or specified by RB-END-PAGE. The default value of RB-BEGIN-PAGE is 0, which means to use the beginning page saved in the report definition. The default beginning page value saved in a report definition is 1.

For example, A user can restart a cancelled report where it was interrupted by specifying the starting page number as the RB-BEGIN-PAGE value and 999999999 as the RB-END-PAGE value.

To reprint one or more consecutive pages of a report, specify the page numbers in the RB-BEGIN-PAGE and RB-END-PAGE parameters. To print just one page, specify the same page number for both parameters.

`RB-DB-CONNECTION`

Allows you to override the saved database connections by specifying the logical name of the database as saved in the report and all connection parameters Report Engine needs to connect to the database containing the report data. You can specify a string composed of any valid Progress 4GL connection parameters.

Report Builder saves all required database connection information with the report definition (except the password). You need to specify a value for RB-DB-CONNECTION only in three situations:

- If you want to run a report using a database other than the one saved with the report definition
- If you want to run a report using the same database that is saved with the report definition, but you are connecting to that database on the command line that invokes Report Engine (or in a .pf file that Report Engine reads on startup)
- If your database requires a password, and you do not want Report Engine to prompt you for it

Use the logical database name, not the physical database name, when specifying which report databases are to be overridden with the RB-DB-CONNECTION parameter.

If you have security measures such as passwords and user IDs enabled, you must specify the Password (-P) and User ID (-U) parameters or the Report Password (-rbP) parameter if you are using the table interface.

### RB-DB-CONNECTION and the Report Engine Table Interface

When you use the Report Engine table interface, you can use RB-DB-CONNECTION in two ways:

- Specify all the database connection information. To do this, simply specify the database connection information using the Progress 4GL connection parameters.
- Specify to use a database to which Report Engine was connected by the command line. To do this, you must specify a complete database connection for the report on the command line or in the *rbstart.pf* file, then specify the RB-DB CONNECTION values. See Chapter 4, "Report Engine Table Interface," for information about using and invoking the table interface, including using the *rbstart.pf* file.

You use two symbols to indicate different types of database connection overrides: “=>” and “=”. The “=>” symbol indicates to use a database already connected on the command line. The “=” symbol indicates to establish a new database connection and ignore any databases connected on the command line. The “=>” override can **only** be used with the table interface.

When you use these symbols to specify the database connection information, follow these rules:

- On the left side of both symbols (=> and =), specify the logical name of a database saved with the report.
- On the right side of the “=>” symbol, specify the logical name of a database that was connected on the command line.
- On the right side of the “=” symbol, specify a complete connection string.

To override a connection to a Progress Atlas database saved in the report definition with a connection to a non-Progress database, you must specify two connections. The first connection is to the Progress schema holder and the second is to the non-Progress database. Because you must use two connections, you cannot simply use the “=” symbol in the RB-DB-CONNECTION parameter. Instead, you must connect to the schema holder and non-Progress database on the command line and then use the “=>” symbol in the RB-DB-CONNECTION parameter to override the database used in the report with the non-Progress database. This can only be done with the table interface.

**NOTE:** If you put the Report Engine table into the same database as your application data, and if you create a report that refers to that database, you must use the RB-DB-CONNECTION parameter with the => symbol. If you do not, Report Engine might indicate that the database is already connected when it tries to run the report.

The problem is that because the Report Engine table information is in the database, you must specify the database connection information on the command line that invokes Report Engine (or in a .pf file that Report Engine reads on startup). This means that Report Engine will already be connected to the database by the time it starts working on the report; when it tries to connect to the database requested in the report, it will be attempting to connect to the same database a second time.

Sometimes, Report Engine considers this an error; however, even if it does not, this second connection can slow down your database server. To avoid this, when you run this report, you must specify a value for RB-DB-CONNECTION that tells Report Engine to reuse the connection it has already made, rather than trying to establish a new connection for the report.

The RB-DB-CONNECTION value looks something like this:

```
logical1 => logical2
```

Note that logical1 is the logical name of the database as stored in the report and logical2 is the logical name used for the same database in the connection parameters on the command line. In this situation, where the Report Engine database is the same as the application database, these two logical names are often the same.

The following example shows how to override the stored connection information. If the report definition specifies the physical database as newdb1.db and the logical database name as newdb1, you override the saved database with a connection established on the command line with the following steps:

1 ♦ Override the saved database connection information using the following value in the RB-DB-CONNECTION parameter:

```
newdb1 => newdb2
```

The "=>" symbol indicates to use a database already connected on the command line. This parameter value means that when Report Engine runs the report, it uses the connection established on the command line and does not attempt to make a connection to the newdb1 database.

2 ♦ Invoke Report Engine with the following command:

```
RUN aderb/_prore.p(false, "-db newdb2 -1 -rbtable Rep-Eng")
```

This command invokes Report Engine with a connection to the newdb2 database. In this example, the newdb2 database contains both the tables required by the report and the table (Rep-Eng) that specifies the RB-DB-CONNECTION parameter value.

Instead of specifying the connection on the command line, you could have used the following value in the RB-DB-CONNECTION parameter in the Rep-Eng table:

```
newdb1 = -db newdb3 -1
```

The "=" symbol indicates to establish a new database connection using the specified connection information and ignore the databases connected on the command line. This value means that when you invoke Report Engine, it will override the stored database connection information for newdb1 with the information "-db newdb3 -1".

You can override the database connection information for multiple databases using the RB-DB-CONNECTION parameter. To override multiple databases in the same RB-DB-CONNECTION parameter value, separate each section with a new-line character (~n) as shown in the following example:

```
sample => newdb2~ndemo = -db newdb3 -1~nsports = -db newdb4 -1
```

This example overrides the connection information for three report databases (with logical names sample, demo, and sports). Notice that each database override is separated by a new-line character (~n). When entering data into a Progress fill-in field, you must use CHR(10) to identify the new-line character.

When you are overriding database connection information for a report that uses multiple databases, you do not have to provide override information for all the databases; only the ones that you want to override.

There are several reasons to use the table interface with RB-DB-CONNECTION:

- You can specify overrides for more than one database.
- You can use the RB-DB-CONNECTION parameter to reference a database connected on the command line, rather than hard-coding the connect string in the RB-DB-CONNECTION parameter.
- If all the reports are run against the same database, you can connect to it on the command line, and then Report Engine maintains that one database connection for all reports instead of re-establishing the connection as it runs each report.

### RB-DB-CONNECTION and the Report Engine PRINTRB and PRNTRB2 Interfaces

When you use the PRINTRB or PRNTRB2 interfaces to the Report Engine, the RB-DB-CONNECTION parameter allows you to override only one database.

In versions of Report Builder before Version 8.0, the parameters in the RB-DB-CONNECTION parameter to PRINTRB do not completely substitute for the full database connection string in the report definition. Instead, Report Engine substitutes them on a connection-parameter-by-connection-parameter basis. Thus, if your report definition specifies the connection information "-db orders -N tcp -H server -S ordersrvc", and the RB-DB-CONNECTION parameter specifies the string "-H newserver", Report Engine uses the connection string "-db orders -N tcp -H newserver -S ordersrvc" when it attempts to establish the database connection. Therefore, it unintentionally prevents you from overriding the previously described report database connection with a new connection string such as "-db c:\mydata\orders -1", because there is no way to eliminate the -H, -S, and -N connection parameters.

Because some users might have utilized this behavior, Report Builder Version 8.0 and greater retains it. However, Report Builder also augments the syntax of the RB-DB-CONNECTION parameter to allow you to completely replace the report database connection information. If you specify the RB-DB-CONNECTION parameter as "orders=-db c:\mydata\orders -1", where the leading "orders=" indications the logical name of the database as specified in the report definition, Report Engine discards all the connection information saved in the report definition and uses only the newly specified database connection parameters.

With the PRINTRB or PRNTRB2 interfaces, you cannot use the "=>" symbol. However, you can use only the "=" symbol. The "=" symbol indicates to establish a new database connection using the specified connection information.

When you use the "=" symbol to specify the database connection information, follow these rules:

- On the left side of the "=" symbol, specify the logical name of the database saved with the report.
- On the right side of the "=" symbol, specify a complete connection string.

For example, if the report definition specifies the physical database as newdb1.db and the logical database name as newdb1, you use the RB-DB-CONNECTION parameter to override the connection information using the following value:

```
newdb1 = -db newdb3 -1
```

This value means that when you invoke Report Engine, it will override all the stored database connection information for newdb1 with the information "-db newdb3 -1". The "=" symbol indicates to establish a new database connection using the specified connection information.

For backward compatibility, you can still use the syntax required by earlier versions of Report Builder to override individual parameters. For example, the following value tells Report Engine to simply add the values to the existing connection information stored in the report:

```
-db newdb3 -S newserver
```

RB-DISPLAY-ERRORS

Controls whether Report Engine displays an error message dialog box when it encounters an error.

If the parameter is false and you are using the PRINTRB and/or PRNTRB2 interfaces, Report Engine does not display the error messages, but writes them to the report status file in the current working directory. However, if you are using the table interface, Report Engine writes to the report status file only if you use the Report Status File (-rbstatfile) parameter. The *report status file* is a text file to which Report Engine writes status and error information. For information about the status file, see the "Verifying Report Status" section of Chapter 2, "Report Engine Administration."

If the parameter is true, Report Engine displays the error message on the screen in addition to possibly writing it to the report status file. When Report Engine displays the error message dialog box, the user must choose OK to acknowledge the error. Report Engine then either terminates processing of that report or displays a dialog box where the user can enter the correct information.

RB-DISPLAY-STATUS

Controls whether Report Engine displays a Print Status window while it generates a report.

If the parameter is false, Report Engine does not display a Status window, but displays as an icon while it is running.

If the parameter is true, Report Engine displays a Status window.

RB-END-PAGE

Specifies the ending page number for the report. The default value for this parameter is 0, which means to use the ending page saved in the report definition. The default ending page saved in the report definition is 999999999. The value you specify for RB-END-PAGE must be greater than or equal to the report's beginning page, which can be either saved in the report definition or specified using RB-BEGIN-PAGE.

For example, to print only the first ten pages of a long report, specify RB-BEGIN-PAGE as 1 and RB-END-PAGE as 10.

To reprint one or more consecutive pages of a report, specify the page numbers in the RB-BEGIN-PAGE and RB-END-PAGE parameters. To print just one page, specify the same page number for both parameters.

RB-FILTER

Specifies a logical expression to override the filter saved in the report definition, if any, when RB-INCLUDE-RECORDS is set to override ("O").

The syntax of the RB-FILTER expression is identical to that of any calculated field expression that returns a logical value. The RB-FILTER expression can be up to 1024 characters long. When an expression is specified, Report Engine selects all records where the value of the RB-FILTER expression is true. The expression can reference any data, aggregate fields, or calculated fields that are available in the report. See the *Progress Report Builder User's Guide* for the restrictions on filtering with aggregate and calculated fields.

If the expression you enter is not server-evaluatable, the report might run significantly slower. See the *Progress Report Builder User's Guide* for more information about server-evaluatable expressions.

For example, if you enter the filter expression **City = "Dallas"**, Report Engine selects all records where the value in the City field is Dallas. If the city name were in a character field named City-Note, the following filter expression selects all records in which the City-Note field begins with the word "Dallas":

```
"City-Note MATCHES ~"Dallas*~""
```

You must enclose literal strings in quotation marks and use the tilde-quote (~"*string*~") or quote-quote (""*string*"") around the embedded character strings.

Entering **Balance >= 200** selects all records where the value in the Balance field is greater than or equal to 200. To select records where the date in the Order-Date field of the Order table is January 31, 1994, use this format:

```
"Order.Order-Date = 01/31/94"
```

If the same field name exists in more than one table used in your report, you must specify the table alias and the field name as shown in the previous example.

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

```
"(City = ""Dallas"" OR City = ""Houston"") AND Credit-Limit > 50000"
```

See the *Progress Report Builder User's Guide* for information about the order in which Report Builder evaluates expressions.

**NOTE:** You must set RB-INCLUDE-RECORDS to override ("O") for the RB-FILTER to override the filter in the report definition.

`RB-INCLUDE-RECORDS`

Controls whether to apply a filter to the report at run time. Table 3–2 lists the valid RB-INCLUDE-RECORDS values.

**Table 3–2: RB-INCLUDE-RECORDS Values**

| Value | Meaning |
|---|---|
| S | (Saved) runs the report using the filter saved in the report definition, if any. Report Engine ignores the expression in RB-FILTER and runs the report as specified in the report definition. |
| E | (Entire) runs the entire report, ignoring any filter specified in the report definition or in the RB-FILTER parameter. |
| O | (Override) overrides the filter specified in the report definition with the expression in the RB-FILTER parameter. |
| ? | (Question mark) displays a dialog box allowing the user to enter a filter expression or edit the filter specified in the report definition. |
| blank | Runs the report using the filter saved in the report definition, if any. Report Engine ignores the expression in RB-FILTER and runs the report as specified in the report definition. |

When you use the question mark (?) value for the RB-INCLUDE-RECORDS parameter, Report Engine ignores the value of RB-FILTER.

RB-MEMO-FILE

Contains the name and optional directory location of a memo file to be used in place of the one saved in the report definition. The override memo file must be on the machine where Report Engine is running. Report Engine searches for the memo file in the following ways:

| If you specify a . . . | Then Report Engine searches . . . |
|---|---|
| Directory and a filename | Only for the specified directory and file |
| Directory without a filename | The specified directory for the memo file name saved in the report definition |
| File name without a directory | For a file with the specified name in the directory saved in the report definition |

If you leave this parameter blank, Report Engine uses the memo file specified in the report definition.

RB-NO-WAIT

Specifies whether to use synchronous or asynchronous processing when you are using the PRINTRB or PRNTRB2 interfaces. If this parameter is "No", then `aderb/_printrb.p` does not return until Report Engine finishes printing the report. If it is "Yes", then `aderb/_printrb.p` returns as soon as Report Engine starts, allowing the 4GL program to continue with other tasks while the report prints.

RB-NO-WAIT is required only with the PRINTRB and PRNTRB2 interfaces. In order to run multiple instances of Report Engine with PRNTRB2, RB-NO-WAIT must be set to "Yes." You cannot use this parameter if you are using the table interface. Instead, you specify this functionality as an argument to `_prore.p`. See Chapter 4, "Report Engine Table Interface," for more information.

RB-NUMBER-COPIES

Specifies the number of copies of the report you want to print. The number must be between 0 and 999, inclusive. If you enter 0, Report Engine prints the number of copies specified in the report definition. If you specify a number, the specified value overrides the number entered in the Report Builder Print Setup dialog box.

`RB-OUTPUT-FILE`

Saves report output as a file for printing later, or use it in conjunction with RB-PRINT-DESTINATION to export a report to a text file without printer control codes. The name of the output file can include a path. The override output file must be on the machine where Report Engine is running.

For example, to send a report to a text file `Order.txt` in the `c:\projects\text` subdirectory, specify the following value for the RB-OUTPUT-FILE parameter, and RB-PRINT-DESTINATION as "A":

```
c:\projects\text\Order.txt
```

If RB-OUTPUT-FILE does not include a path, Report Engine places the file in the current working directory.

When you specify this parameter, and if RB-PRINT-DESTINATION contains the value **A**, the report outputs as a text file. If the value of RB-PRINT-DESTINATION is anything other than **A**, Report Engine outputs the report to a file with printer control codes.

If RB-OUTPUT-FILE does not include a path, Report Engine places the file in the current working directory.

`RB-PRINT-DESTINATION`

Specifies to print the report on the printer specified in the report definition or with the RB-PRINTER-NAME parameter or to display the report on the screen. Leave the parameter blank to print to the printer saved in the report definition or to the printer specified as the RB-PRINTER-NAME value. Table 3–3 describes the valid RB-PRINT-DESTINATION values.

**Table 3–3: RB-PRINT-DESTINATION Values**

| Value | Meaning |
|---|---|
| D | Displays the report on the screen. This allows the user to preview the report before printing it. After previewing the report, the user can choose Print on the Preview screen to send the report to the printer specified in the report definition or specified as the RB-PRINTER-NAME value. If the value of RB-PRINT-DESTINATION is D and you specify RB-OUTPUT-FILE, when the user chooses Print in Preview, Report Engine outputs the report to the file specified in RB-OUTPUT-FILE, using control codes for the printer specified with RB-PRINTER-NAME. |
| A | Sends the report to the text file named as the RB-OUTPUT-FILE value. Report Engine exports the report as a text file without printer control codes. |
| ? | Allows the user to select the print destination (screen or printer) at run time. When the value of RB-PRINTER-NAME is a question mark, Report Engine prompts the user to also choose the printer at run time. If you specify RB-WINDOW-TITLE, the dialog box title bar contains the RB-WINDOW-TITLE value. If RB-WINDOW-TITLE is empty, the title bar displays the report name. |
| blank | Sends the report directly to the printer specified in the report definition or specified by RB-PRINTER-NAME, or if RB-OUTPUT-FILE is specified to that file. Report Engine includes the printer control codes in the output file for the specified printer, or if no printer is specified, for the saved printer. |

`RB-PRINTER-NAME`

Specifies the printer on which to print the report. You can specify:

- The name of an available Windows printer (for example, “HP LaserJet Series III”). In Report Builder choose Report→ Print Setup to access the Printer Setup dialog box. This dialog box lists the available Windows printers.
- A question mark (?) to allow the user to select a printer at run time. When RB-PRINTER-NAME contains a question mark, the Printer Setup dialog box appears.
- A blank value to use the printer saved in the report definition

Regardless of which value is specified, the printer must be a printer that is available to the machine where Report Engine is running.

When you specify a question mark (?), the Printer Setup option in the Windows Control Panel controls which printers appear in the Printer Setup dialog box. The selected printer is always the printer specified in the report definition. The user can select another printer and port or choose the Setup button to change other print characteristics.

You cannot override the paper tray selection with the Report Engine parameters. You can change the paper tray selction only when running a report. However, if you specify a tray (upper or lower) and use a printer that has only a single tray, Report Engine runs the reports correctly.

`RB-PRINTER-PORT`

Specifies the printer port. You can:

- Specify a port, such as “LPT1:”. You must include the colon (:).
- Enter a question mark (?) value. When RB-PRINTER-PORT contains a question mark, the user sees the Printer Setup dialog box.
- Specify a blank value to use the printer port saved in the report definition.

Regardless of which value is specified, the port must be on the machine where Report Engine is running.

`RB-REPORT-LIBRARY`

Identifies the library that contains the report. The library name can include a path. If you do not include a path, Report Engine searches for the library in the default directory specified in the registry or `progress.ini` . If you have not specified a default library directory, Report Engine searches for the library in the current working directory. The .prl extension is optional. The report library must be accessible from the machine on which Report Engine is running.

**NOTE:** You must specify the library name. If you leave this parameter blank or if the library you specify cannot be found or read, Report Engine writes an error in the status file or RB-STATUS field, depending on the interface or optional command-line parameters you are using. Regardless of which interface you use, if you enable RB-DISPLAY-ERRORS, Report Engine displays an error message box.

For example, a value of `c:\projects\reports` identifies the report library as `reports.prl` in the subdirectory `projects` on drive C.

RB-REPORT-NAME

Specifies the report name.

For example, to run a report named Discount Order, enter "Discount Order" as the value for this parameter. The report you specify must be in the library specified by RB-REPORT-LIBRARY.

**NOTE:** You must specify the report name. If you leave this parameter blank or if the report does not exist in the library you specify, Report Engine writes an error in the status file or RB-STATUS field, depending on the interface or optional command-line parameters you are using. Regardless of which interface you use, if you enable RB-DISPLAY-ERRORS, Report Engine displays an error message box.

RB-STATUS

Specifies the status of the report generation for the table interface. If you specify the Report Update (-rbupds) parameter when you invoke Report Engine, Report Engine updates the RB-STATUS field with information describing where it is in the report generation process. Table 3–4 describes the possible parameter values.

**Table 3–4: RB-STATUS Values**

| Value | Meaning |
|---|---|
| STARTED | Report Engine has started generating the report. |
| COMPLETED | Report Engine has finished generating the report. This does not mean the report has been printed. |
| ERROR: *message* | Report Engine has detected an error and cannot complete generating the report. |
| blank | The Report Update (-rbupds) parameter is not set or Report Engine has not yet started generating this report. |
| CANCELLED | The user selected to cancel the report. |

For example, if you specify -rbupds when you invoke Report Engine, when Report Engine starts generating the report, it sets the value of the RB-STATUS field to “STARTED.”

You optionally use RB-STATUS when you use the table interface. You cannot use this parameter with the PRINTRB interface.

RB-STATUS-FILE

Specifies the filename of the report status file into which Report Engine will write the status information.

This parameter tells Report Engine to create a file containing information about the report it has generated. This file is called the report status file. If the file does not exist, Report Engine creates one. If the file does exist, Report Engine overwrites it. If you do not specify a filename, Report Engine does not generate a status file. If you do not specify the full pathname, Report Engine uses its current working directory.

For example, to create a status file `status.txt` in the `c:\work\status` subdirectory, specify the following value for the RB-STATUS-FILE parameter:

```
c:\work\status\status.txt
```

RB-TAG

Specifies a tag that determines whether to generate a report, based on the value from the Report Tag (-rbtag) parameter, when you use the table interface. When Report Engine generates the reports for which you have specified report parameters in the table, it generates only those whose values in this parameter match the -rbtag parameter value.

If you do not specify a -rbtag value, Report Engine runs all the reports in the Report Engine table, regardless of the value in the RB-TAG field.

For example, if the Report Engine table contains report records for four different quarterly reports, you can assign each set of records for a quarter a different RB-TAG value such as Q1, Q2, Q3, and Q4. Then when you invoke Report Engine, specify the desired tag with the -rbtag parameter. For example, to run the reports for the first quarter, you specify Q1 as the -rbtag value:

```
RUN aderb\_prore.p(false, "-db sample.db -1 -rbtable Rep-Eng
       -rbtag Q1")
```

Another way to use this parameter is to store the reports for multiple users in the Report Engine table, then specify a specific user's tag to run only his or her reports.

You optionally use RB-TAG when you use the table interface. You cannot use this parameter with the PRINTRB or PRNTRB2 interfaces.

RB-TEST-PATTERN

Controls whether Report Engine allows the user to print a test pattern. A test pattern is useful for aligning forms or labels in the printer.

The test pattern flag can be either true or false. True means to display a prompt before printing the report to allow the user the option of printing a test pattern. False means do not display this prompt.

If the parameter is false, Report Engine does not offer a choice to print a test pattern.

If the parameter is true, Report Engine prompts the user to select whether to print the test pattern before printing the report. The user can print the test pattern as many times as necessary and then print the report.

RB-WINDOW-TITLE

Displays the specified title in the following places:

- The Title Bar of the Preview window
- The Print Status window (if RB-DISPLAY-STATUS = True)
- Below the Runtime icon (if RB-DISPLAY-STATUS = False)
- The title bar of the dialog box that displays when RB-PRINT-DESTINATION = ?

By default, Report Engine displays the report name in these places. You can enter up to 31 characters for the window title.

## 3.3 User-defined Parameters

*User-defined parameters* are character string arguments that you specify in the RB-OTHER-PARAMETERS field. Regardless of which interface you use, you can specify multiple user-defined parameters or arguments, but Report Engine expects to find them all in a single RB-OTHER-PARAMETERS value.

User-defined parameters provide only character values (text strings) to Report Engine. If you want to access a numeric, date, or logical value, you must perform the conversion from character to the desired data type within the report definition. See the "Overriding a Table Name" section for information about converting values.

You can use user-defined parameters to provide information that you do not want hard-coded in the report definition. You can also specify user-defined parameters to override table specifications and prompt for user input.

The following steps describe how to override all report characteristics except table names. See the "Overriding a Table Name" section for an example of how to specify different table names for a report. Follow these steps to use a user-defined parameter:

1 ♦ Decide upon a parameter name to identify the parameter. Parameter names can be up to 32 characters long and can consist of alphabetic characters (A–Z, a–z), digits (0–9), and special characters ($, &, #, %, –, _). In addition, parameter names must begin with a letter (A–Z, a–z). Each name must be unique within the report and cannot contain spaces.

2 ♦ In the Report Builder, create a calculated field that uses the RUNTIME-PARAMETER( ) function using the following format for the field's expression:

```
RUNTIME-PARAMETER("parameter-name")
```

The RUNTIME-PARAMETER( ) function takes a quoted parameter name as its argument and returns the value of the parameter as a string.

3 ♦ Place the calculated field in the report layout.

4 ♦ When you run Report Engine, enter the following code for the RB-OTHER-PARAMETERS value:

```
parameter-name = value
```

If you allow your users to specify report characteristics at run time, you can create one report definition and use it with the user input to run many different reports from the same report definition. See the "Prompting for User Input" section for information about prompting users for information.

This section describes the following examples of how to use user-defined parameters:

- Overriding a table name
- Specifying an author's name
- Specifying multiple user-defined parameters
- Overriding the sort criteria

### Overriding a Table Name

You can override the tables specified in the report definition with tables that have the same structure. Therefore, if you store your quarterly information in four tables with identical structures and you create the report definition with the First-Quarter table, when you run Report Engine, you can use the Second-Quarter, Third-Quarter, or Fourth-Quarter table instead.

The syntax for overriding tables is different from the generic user-defined parameter syntax in that the parameter "name" is a report "table-alias". To override the table specified in the report definition, enter the table alias and the new table name.

For example, if you specify the First-Quarter table in the report definition with an alias of "Quarter", when you run Report Engine, enter the following code in the RB-OTHER-PARAMETERS parameter to use the Third-Quarter table instead:

```
"Quarter = Third-Quarter"
```

### Specifying an Author's Name

You can specify an author's name when you generate a report. For example, to print the report author's name at the top of the report, follow these steps:

1 ♦ In Report Builder, create a calculated field named My-Name with the following expression:

```
RUNTIME-PARAMETER("My-Name")
```

2 ♦ Place My-Name in a title band line.

3 ♦ When you run Report Engine, specify the author's name by entering the following as the RB-OTHER-PARAMETERS value:

```
"My-Name = Todd Winmill"
```

Report Engine prints "Todd Winmill" in the title area of the report.

### Specifying Multiple User-defined Parameters

You can specify up to 99 table override parameters and as many user-defined parameters as you need. Follow these rules when entering multiple parameters:

- Separate each parameter with a 1-byte linefeed character (ASCII value 10.) In character string constants in 4GL code, you can use the two-character symbol "~n" to represent the 1-byte linefeed character.
- Use only one set of quotation marks for the entire argument. Because all of the parameters make up one character string, you cannot enclose each parameter in a separate set of quotation marks.

For example:

```
"City = Boston~nState = MA~nQuarter = Third-Quarter"
```

**NOTE:** The two-character sequence "~n" represents the linefeed character only when you use it with a 4GL application. If you enter "~n" in a Progress fill-in field, Progress does not convert it to a 1-byte linefeed character.

Another way to separate multiple user-defined parameters is to use "CHR(10)" to identify a line feed instead of embedding "~n" in a string.

For example:

```
"City = Boston" + CHR(10) + "State = MA" + CHR(10) + "Quarter =
Third-Quarter"
```

Notice that when you use "CHR(10)" instead of "~n", you must enclose each user-defined parameter in quotation marks.

### Overriding the Sort Criteria

You can override the sort criteria at run time with a user-defined parameter. For example, to specify the sort criteria, follow these steps:

1 ♦ In Report Builder, create a calculated field named Sort-Criteria with the following expression:

```
CASE(RUNTIME-PARAMETER("SORTFIELD"), "NAME", Name, "STATE", State,
  "ZIP CODE", Postal-Code, City)
```

Because the CASE function requires that all potential return values have the same data type, you must list fields that have the same data type for sort criteria options.

2 ♦ Sort on the Sort-Criteria field.

3 ♦ When you run Report Engine, choose a sort criteria from the four choices (Name, State, Postal-Code, or City), using the RB-OTHER-PARAMETERS parameter. For example:

```
"SORTFIELD = NAME"
```

**NOTE:** If you override the sort order at run time, the sort will always be evaluated by Report Builder.

## 3.3.1 Parameter Guidelines and Restrictions

Each time you run Report Engine, you must specify a value for each of its parameters, even if that value is blank or zero (for the PRINTRB and PRNTRB2 interfaces) or is the parameter field's default value (for the table interface). Each of the parameter values must be of the correct data type for the corresponding Report Engine parameter.

**NOTE:** For the PRINTRB and PRNTRB2 interfaces, the parameters must also be in the correct order (the order in which they appear in Table 3–1).

You must specify a value for two parameters: RB-REPORT-LIBRARY and RB-REPORT-NAME. If you do not specify the report name and library name, Report Engine cannot run the report. For the remaining parameters, you can specify either a real value or a special "default" value. The "default" value specifies to use the value stored in the report definition. The "default" value is different for each data type.

**Table 3–5: Report Engine Parameter Default Values**

| Parameter Data Type | Default Value |
|---|---|
| Character | "" |
| Numeric | 0 |
| Logical | NA |

When you use the table interface, the default values listed in Table 3–5 match the Initial Value that Progress uses by default when you create a new record in the Report Engine table. (The Initial Value is specified when you create a field in the Data Dictionary.) Thus, when you create a record, all the fields are set to the default value . However, you must assign actual values to RB-REPORT-NAME and RB-REPORT-LIBRARY fields before you call Report Engine.

### 3.3.2 Prompting for User Input

You can use both predefined and user-defined parameters to prompt users for input. However, the user must be using the machine where Report Engine is running.

#### Prompting for Input with a Predefined Parameter

There are two ways to prompt for user input using predefined parameters. The first way is to write your own 4GL code to prompt users for input, then pass the values to Report Engine using predefined parameters. The second way to prompt for user input is to use the built-in Report Engine prompting mechanism. Several of the predefined parameters allow you to specify a question mark (?) as the parameter value. When you specify a question mark, Report Engine automatically prompts the user for input for the parameter.

You can specify a question mark for the RB-PRINT-DESTINATION, RB-PRINTER-NAME, RB-PRINTER-PORT, and RB-INCLUDE-RECORDS parameters.

#### Prompting for Input with a User-defined Parameter

You can prompt a user for other report information using a user-defined parameter.

In the “User-defined Parameters” section, the example shows how to print the report author’s name at the top of the report. To prompt the user to enter his or her name, follow these steps:

1 ♦ In the Report Builder, create a calculated field named MYNAME with the following expression:

```
RUNTIME-PARAMETER("MYNAME")
```

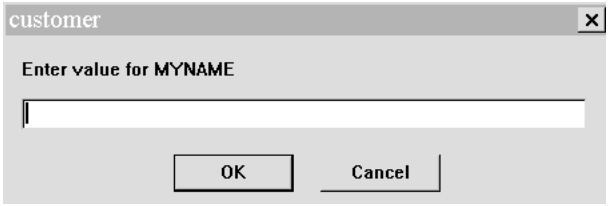
2 ♦ Place MYNAME in the title band line.

3 ♦ When you run the Report Engine, you can:

- Use the Report Engine prompt to prompt the user to enter his or her name using the following code in the RB-OTHER-PARAMETERS parameter:

```
"MYNAME = ?"
```

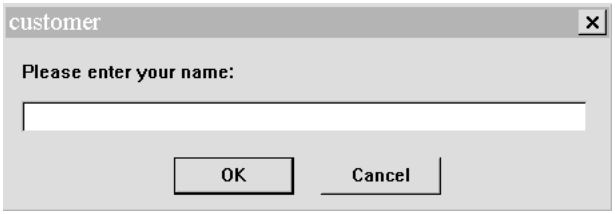
This causes the Report Engine Parameter dialog box shown below to appear:




- Customize the prompt using the following code:

```
"MYNAME = ?Please enter your name:"
```

The Report Engine Parameters dialog box will appear as shown below:




When you prompt users for input, the Report Engine Parameter dialog box title bar contains the value of the RB-WINDOW-TITLE parameter. If RB-WINDOW-TITLE is empty or UNKNOWN, Report Engine uses the report name as the dialog box title.

You cannot modify the dimensions of the dialog box to accommodate the anticipated user input. The dialog box always remains the same size and users can enter up to 512 characters. If the user chooses the Cancel button, Report Engine does not run the report and writes the "Cancelled" message to the report status file or the RB-STATUS field, depending on the interface and command-line parameters you use. For information about the status file, see the "Verifying Report Status" section of Chapter 2, "Report Engine Administration."

### 3.3.3 Incorporating Parameters in the Report Definition

The RUNTIME-PARAMETER() function takes a parameter name as its argument and returns the value of the parameter as a string. This means that user-defined parameters provide only character values (text strings) to Report Engine. If you want to access a numeric, date, or logical value, you must perform the conversion from character to the desired data type within the report definition.

To convert to numeric, date, or logical values, use the functions shown in the following table in a calculated field in the report definition:

| To convert to . . . | Use . . . |
|---|---|
| Numeric data | NUMERIC (*parameter*) |
| Dates | DATE (*parameter*) |
| Logical data | IIF (*parameter* = "Yes", Yes, No) |

You can also use the RUNTIME-PARAMETER( ) function to get the value of predefined parameters whose arguments are character strings.

For example, to print the database connection information on the title band line, follow these steps:

1 ♦ In Report Builder, create a calculated field named Connect-String with the following expression:

```
RUNTIME-PARAMETER("RB-DB-CONNECTION")
```

2 ♦ Place Connect-String in the title band line.

When Report Engine prints the report, it displays the database connection information specified by the RB-DB-CONNECTION parameter in the report title area.

# 4

# Report Engine Table Interface

When you use the Report Engine table interface, you store all the Report Engine parameter values in a database table. This chapter describes:

- Table interface basics
- Table interface quick start example
- Creating the Report Engine table
- Entering the Report Engine parameter values in the table
- Invoking Report Engine
- Running multiple reports
- Running reports from an application

## 4.1 Table Interface Basics

The Report Engine table interface stores the report parameters in a database table called the *Report Engine table*, then uses the parameters to generate one or more reports. The fields in the Report Engine table that store the report parameters are called *Report Engine fields*. These fields contain values for the required and optional report parameters. A *Report Engine record* is a record that contains the Report Engine fields.

Being able to store report information in the Report Engine table allows you to run batches of reports without running the Progress 4GL. Another advantage of using the table interface instead of the PRINTRB or PRNTRB2 interface is that the table interface allows you to invoke Report Engine directly from the command line or a Windows icon.

Follow these general steps to use the table interface:

1. Create the Report Engine table.
2. Create a record in the table for each report and assign values to some of the Report Engine fields in each record.
3. Invoke Report Engine to generate the reports.

## 4.2 Table Interface Quick Start Example

As described in the "Table Interface Basics" section, there are three steps you must perform to use this interface. The first step in this example is to create the Report Engine table in a new or existing database. Progress Software provides the `rbreport.df` definition file to make getting started with the interface easier. The file contains loadable definitions for the RBREPORT table. The second step is to create a record in the table and set some of the fields in that record, and the third step is to invoke Report Engine. The `rbstart1.p` procedure performs the second and third tasks as an example.

Follow these steps to create the Report Engine table using the `rbreport.df` file, and then invoke Report Engine using it:

1 ♦ Create a database called Runtable containing the Report Engine table. You can use the database definition file (`%DLC%\bin\rbreport.df`) that contains loadable definitions for the RBREPORT table. (Note that you can give the database any name you choose, and it can contain other tables, such as the tables that contain your report data.)

2 ♦ Create an MS-Windows icon with the following command on the command line:

```
wproserv -db dbname
```

Note that *dbname* contains the full pathname of the Runtable database.

3 ♦ Double-click the MS-Windows icon to start the database server. The icon becomes minimized.

4 ♦ Start Progress and connect to the Runtable database in multi-user mode.

5 ♦ Start the Procedure Editor.

6 ♦ Run the `rbstart1.p` procedure.

Here is the code for the `rbstart1.p` procedure:

### rbstart1.p

```
  DO TRANSACTION:

/* 1 */
  CREATE RBREPORT.

/* 2 */
  ASSIGN
    RBREPORT.RB-REPORT-LIBRARY = "c:\dlc\src\aderb\rbsample.prl"
    RBREPORT.RB-REPORT-NAME = "Customer Discount"
    RBREPORT.RB-PRINT-DESTINATION = "D"
    RBREPORT.RB-DISPLAY-STATUS = yes
    RBREPORT.RB-DISPLAY-ERRORS = yes.

  RELEASE RBREPORT.

  END.

/* 3 */
  RUN aderb\_prore(false, "-db Runtable -S servername -H hostname
    -N networktype -rbdel").
```

The commented numbers correspond to the following step-by-step descriptions:

1. Create an RBREPORT record.

2. Enter the values for five of the record fields. Report Engine uses the default values for the fields for which you do not specify a value.

3. Call the `_prore.p` procedure to invoke Report Engine and specify the *NO-WAIT-value* and the database connection information. By specifying "false" as the *NO-WAIT-value*, you instruct Progress to wait until Report Engine completes processing before continuing with the application. Finally, the Report Delete (-rbdel) parameter deletes the record from the Report Engine table when the procedure is done.

   See the "Methods of Invoking Report Engine" section for more information about the `_prore.p` procedure.

## 4.3 Creating the Report Engine Table

Before you can use the table interface to generate reports, you must create the Report Engine table in which to store the report parameters. Table 4–1 lists the field names and data types that you must specify when you create the Report Engine table.

**Table 4–1: Required Field Names and Data Types** *(1 of 2)*

| Field Name | Contents | Data Type |
|---|---|---|
| RB-REPORT-LIBRARY | Report library name | CHARACTER |
| RB-REPORT-NAME | Report name | CHARACTER |
| RB-DB-CONNECTION | Database connection override string | CHARACTER |
| RB-INCLUDE-RECORDS | Filter flag | CHARACTER |
| RB-FILTER | Filter expression | CHARACTER |
| RB-MEMO-FILE | Memo filename | CHARACTER |
| RB-PRINT-DESTINATION | Print destination | CHARACTER |
| RB-PRINTER-NAME | Printer name | CHARACTER |
| RB-PRINTER-PORT | Printer port | CHARACTER |

**Table 4–1: Required Field Names and Data Types** *(2 of 2)*

| Field Name | Contents | Data Type |
|---|---|---|
| RB-OUTPUT-FILE | Output filename | CHARACTER |
| RB-NUMBER-COPIES | Number of copies | INTEGER |
| RB-BEGIN-PAGE | Beginning page number | INTEGER |
| RB-END-PAGE | Ending page number | INTEGER |
| RB-TEST-PATTERN | Test pattern flag | LOGICAL |
| RB-WINDOW-TITLE | Window title | CHARACTER |
| RB-DISPLAY-ERRORS | Display error flag | LOGICAL |
| RB-DISPLAY-STATUS | Display status flag | LOGICAL |
| RB-OTHER-PARAMETERS | User-defined parameters | CHARACTER |

Notice that each Report Engine field contains one predefined Report Engine parameter. In addition to the Report Engine fields listed in Table 4–1, you can optionally specify the fields described in Table 4–2.

**Table 4–2: Optional Field Names and Data Types**

| Field Name | Contents | Data Type |
|---|---|---|
| RB-TAG | Which reports to generate based on the Report Tag (-rbtag) parameter value | CHARACTER |
| RB-STATUS | Status of the report generation | CHARACTER |

You can also specify other fields in the Report Engine table. However, Report Engine ignores any fields other than the ones listed in Table 4–1 and Table 4–2.

Progress Software provides a database definition file (`%DLC%\bin\rbreport.df`) containing loadable definitions for the Report Engine table interface. You can use this definition file to create a Report Engine table.

If any of the required Report Engine fields is missing when you invoke Report Engine, it will discontinue processing all of the reports. If the command-line parameters allow Report Engine to report errors, it also returns an error message to the report status file or the RB-STATUS field. See Chapter 2, "Report Engine Administration," for a complete description of the report status file.

**NOTE:** Note that in the provided database definition file, the Report Engine table is defined as a hidden table. This means that it will not appear in any table lists unless you specify to display hidden tables.

### 4.3.1 Specifying Default Field Values

When you create the Report Engine table, be sure to set the initial value of each field to the default value that Report Engine expects. By default, when you create fields in the Data Dictionary, Progress assigns the initial values listed in Table 4–3.

**Table 4–3: Data Dictionary Default Initial Field Values**

| Field Data Type | Initial Value |
|---|---|
| CHARACTER | empty string |
| INTEGER | 0 |
| LOGICAL | No |

These initial values correspond to the expected Report Engine default values except for the RB-REPORT-LIBRARY and RB-REPORT-NAME fields. Because you must supply a value for both of these fields, Report Engine cannot accept the default value of an empty string.

## 4.4 Entering the Report Engine Parameter Values for the Reports

After you create the Report Engine table, use an application to input the values for the reports into the table. After you input the report parameter values into the Report Engine table, you can invoke Report Engine.

## 4.5 Invoking Report Engine

You can invoke Report Engine either from the command line or from within an application. However, when you invoke Report Engine from an application, you are simply using the command-line syntax and parameters in a 4GL application.

This section describes:

- Command-line syntax
- Command-line parameters
- Methods for invoking Report Engine

When you invoke Report Engine you should also consider whether you want to be able to verify the report processing status and whether you want to employ password security. InChapter 2, "Report Engine Administration," see the "Verifying Report Status" section for more information about verifying report status, and see the "Using Password Security" section for more information about security considerations.

### 4.5.1 Command-line Syntax

When you invoke Report Engine, you use the `prore32.exe` executable with optional command-line parameters. This is the basic syntax for invoking Report Engine:

**SYNTAX**

```
prore32.exe db-connection [ parameters ]
```

See the "Command-line Parameters"section for a complete description of available command-line parameters.

### 4.5.2 Command-line Parameters

You can use all the standard Progress 4GL startup parameters, including the Parameter File (-pf) parameter, plus some new parameters that are specific to Report Engine.

Table 4–4 describes the locations where you can specify startup parameters.

**Table 4–4: Valid Startup Parameter Locations**

| Location | Description |
|---|---|
| *startup.pf* file | The *startup.pf* file is the parameter file at which all Progress clients look when they start. Therefore, Report Engine scans this, and any parameter files referenced in the file, looking for parameters. The PROSTARTUP environment variable contains the name of the startup.pf file. If PROSTARTUP is not defined, the default file name is "startup.pf."<br><br>Because all Progress clients reference this parameter file, it cannot contain any Report Engine-specific startup parameters. Also, do not specify database connection parameters. |
| *rbstart.pf* file | The *rbstart.pf* file is an optional parameter file that only Report Builder and Report Engine scan. The RBSTARTUP environment variable contains the name of the startup.pf file. If RBSTARTUP is not defined, Report Builder and Report Engine look in the DLC directory for a file named "rbstart.pf." If they do not find the file there, they give up their search and continue their startup process. To include the RBSTARTUP environment variable in the registry or `progress.ini` file, include it in the [RBStartup] section.<br><br>You can specify both Progress startup parameters and Report Engine-specific parameters in this parameter file. |
| Command line | When you specify parameters on the command line, they override any other parameter values stored in the *startup.pf* file or the *rbstart.pf* file. |

There are many Progress 4GL startup parameters that you might want to use when you invoke Report Engine. For example, you can use the Date Format (-d) parameter to change the display format for dates in reports. See the *Progress Startup Command and Parameter Reference* for a complete listing of the Progress 4GL startup parameters.

Table 4–5 lists the Report Engine startup parameters and their syntax.

**Table 4–5: Report Engine Startup Parameters**

| Parameter | Syntax |
|---|---|
| Report Engine Record Delete | `-rbdel` |
| Report Status File | `-rbstatfile` *filename* |
| Encoded Password | `-rbP` *password* |
| Report Engine Table Name | `-rbtable` *table-name* |
| Report Tag | `-rbtag` *tag* |
| Report Update Status | `-rbupds` |
| Report Code Page In | -rbcpreportin *codepage* |
| UDF Code Page In | -rbcpudfin *codepage* |

The remainder of this section describes each startup parameter, in alphabetical order by operating system syntax.

### Report Code Page In (-rbcpreportin)

| Operating System and Syntax | Windows | -rbcpreportin *codepage* | | |
|---|---|---|---|---|
| **Use With** | **Maximum Value** | **Minimum Value** | **Single-user Default** | **Multi-user Default** |
| - | - | - | - | - |

`codepage`

Name of code page to use for the report.

Allows you to override the code page name stored in any reports you open. When you save a report, Report Builder assigns it a code page. When Report Engine attempts to read the report, the -rbcpreportin parameter tells Report Engine to treat the report as being in the specified code page. If you do not specify a value, Report Engine uses whatever code page was assigned to the report when it was last saved.

### UDF Code Page In (-rbcpudfin)

| Operating System and Syntax | Windows | -rbcpudfin *codepage* | | |
|---|---|---|---|---|
| **Use With** | **Maximum Value** | **Minimum Value** | **Single-user Default** | **Multi-user Default** |
| - | - | - | - | - |

codepage

Name of code page to use for the user-defined function file.

Allows you to override the code page name stored in the user-defined function file (PRORB.UDF). When you define a UDF, Report Builder assigns it a code page. When Report Engine attempts to read a report that uses a UDF, the -rbcpudfin parameter tells Report Engine to treat the UDF as being in the specified code page. If you do not specify a value, Report Engine uses whatever code page was assigned to the UDF when it was last saved.

### Report Engine Record Delete (-rbdel)

| Operating System and Syntax | Windows | -rbdel | | |
|---|---|---|---|---|
| **Use With** | **Maximum Value** | **Minimum Value** | **Single-user Default** | **Multi-user Default** |
| - | - | - | - | - |

This parameter tells Report Engine to delete the Report Engine table record corresponding to the report when it has finished generating the report. If you do not specify this parameter, Report Engine leaves the record intact.

### Report Status File (-rbstatfile)

| Operating System and Syntax | Windows | -rbstatfile *filename* | | |
|---|---|---|---|---|
| **Use With** | **Maximum Value** | **Minimum Value** | **Single-user Default** | **Multi-user Default** |
| - | - | - | - | - |

`filename`

The pathname of the report status file into which Report Engine will write the status information.

This parameter tells Report Engine to create a file containing information about the reports it has generated. This file is called the report status file. If the file does not exist, Report Engine creates one. If the file does exist, Report Engine overwrites it. If you do not specify the full pathname, Report Engine uses its current working directory. If the Report Engine table contains multiple reports, the output file will contain information about all of the reports.

See Chapter 2, "Report Engine Administration," for a description of the report status file.

### Encoded Password (-rbP)

| Operating System and Syntax | Windows | -rbP *password* | | |
|---|---|---|---|---|
| **Use With** | **Maximum Value** | **Minimum Value** | **Single-user Default** | **Multi-user Default** |
| - | - | - | - | - |

`password`

An encoded password.

This parameter is an optional Report Engine-specific version of the Progress 4GL Password (-P) startup parameter. The -rbP parameter allows you to hide the value of a password.

Report Engine assumes that the value of the Encoded Password (-rbP) parameter was encoded using the `aderb/_rbpwenc.p` and therefore applies the reverse encoding and treats the result as if it had been provided by the Password (-P) parameter. See the "Using Password Security" section in Chapter 2, "Report Engine Administration" for more information about security considerations.

You can use this parameter when you invoke Report Engine on the command line or in place of the -P parameter in the RB-DB-CONNECTION field of the Report Engine record.

### Report Engine Table Name (-rbtable)

| Operating System and Syntax | Windows | -rbtable *table-name* | | |
|---|---|---|---|---|
| **Use With** | **Maximum Value** | **Minimum Value** | **Single-user Default** | **Multi-user Default** |
| – | – | – | RBREPORT | RBREPORT |

table-name

The name of the Report Engine table containing the report parameters.

If you do not specify this parameter, Report Engine assumes the table name is RBREPORT. When you invoke Report Engine, you must specify the connection parameters that tell Report Engine how to connect to the database containing this table. If you connect to more than one database on the command line, it looks for this table in the first database specified.

### Report Tag (-rbtag)

| Operating System and Syntax | Windows | -rbtag *tag* | | |
|---|---|---|---|---|
| **Use With** | **Maximum Value** | **Minimum Value** | **Single-user Default** | **Multi-user Default** |
| – | – | – | – | – |

tag

A character string that indicates which reports to run.

The Report Tag (-rbtag) parameter tells Report Engine which reports to generate. If you do not specify this parameter, Report Engine generates all the reports in the Report Engine table.

For example, if the Report Engine table contains report records for four different quarterly reports, you can assign each set of records for a quarter a different tag such as Q1, Q2, Q3, and Q4 in the RB-TAG field. Then when you invoke Report Engine, specify the desired tag with the -rbtag parameter. To run the reports for the third quarter, you specify Q3 as the -rbtag value. Another way to use this parameter is to store the reports for multiple users in the Report Engine table, giving each record an RB-TAG value that is the corresponding user's name, then specify a specific user's tag to run only their reports.

### Report Update Status (-rbupds)

| Operating System and Syntax | Windows | -rbupds | | |
|---|---|---|---|---|
| **Use With** | **Maximum Value** | **Minimum Value** | **Single-user Default** | **Multi-user Default** |
| – | – | – | – | – |

The Report Update Status (-rbupds) parameter tells Report Engine to update the RB-STATUS field in the Report Engine table for each report.

If you specify the Report Update Status (-rbupds) parameter when you invoke Report Engine, Report Engine updates the RB-STATUS field with information describing where it is in the report generation process.Table 4–6 describes the possible parameter values Report Engine can provide to the RB-STATUS field.

**Table 4–6: RB-STATUS Values**

| Value | Meaning |
|---|---|
| STARTED | Report Engine has started generating the report. |
| COMPLETED | Report Engine has finished generating the report. This does not mean the report has been printed, only that it has been sent to the printer queue. |
| ERROR: *message* | Report Engine has detected an error and cannot complete generating the report. |
| blank | The Report Update Status (-rbupds) parameter is not set or Report Engine has not yet started generating this report. |
| CANCELLED | The user cancelled the report. |

If you do not specify the -rbupds parameter when you invoke Report Engine, Report Engine ignores the RB-STATUS field, and the field remains blank.

### 4.5.3 Methods of Invoking Report Engine

You can invoke Report Engine using several different methods:

- From an application using the _prore.p procedure file
- From an application using the OS-COMMAND statement
- From a DOS window
- From an MS-Windows icon
- From the Start Menu using the Run option

The following sections describe each of these methods.

#### Using the _prore.p Procedure File

The advantage to invoking Report Engine with the _prore.p procedure file is that you can specify to have the application wait until Report Engine completes generating a report before continuing to execute the remainder of the program.

To invoke Report Engine using the _prore.p procedure file, you must create an application that runs the _prore.p file using the following syntax:

**SYNTAX**

```
RUN aderb\ prore.p ( NO-WAIT-value ,
command-line arguments )
```

Note that _prore.p is the name of a Report Engine procedure that invokes prore.exe automatically using the specified parameters. *NO-WAIT-value* specifies whether to wait for Report Engine to complete processing before running the remainder of the application. The c*ommand-line arguments* value is a character that must begin with the database name and connection parameters required to connect to the Report Engine database, followed by any other Progress startup or database connection parameters or Report Engine startup parameters.

**NOTE:** The database connection information, plus all the optional parameters, cannot exceed 114 characters. If you must enter more characters, use a parameter (.pf) file to hold the parameters.

For example, you might use the following statement to invoke Report Engine:

```
RUN aderb\_prore.p(true, "-db sample.db -1 -rbtable Rep-Eng -rbtag Q1")
```

In this statement, “true” instructs Progress **not** to wait until Report Engine completes processing before continuing with the application. “Sample” is the name of the database that contains the Rep-Eng table. The -rbtable parameter specifies that Rep-Eng is the Report Engine table, and -rbtag indicates to generate only the first quarter reports, or in other words, those reports that have a tag of “Q1.”

If you change the NO-WAIT value as shown in the following example, Progress waits until Report Engine completes processing before continuing the application:

```
RUN aderb\_prore.p(false, "-db sample.db -1 -rbtable Rep-Eng -rbtag Q1")
```

### Using OS-COMMAND Statement

When you invoke Report Engine using the OS-COMMAND statement, you must use the NO-WAIT option. This means that you cannot specify whether to have Progress wait for Report Engine to complete processing before continuing the remainder of the application. Therefore, if you require Progress to wait for Report Engine, you cannot use the OS-COMMAND statement and must invoke Report Engine using the `_prore.p` file.

Use the following command syntax to invoke Report Engine:

#### SYNTAX

```
OS-COMMAND NO-WAIT prore32.exe db-connection [ parameters ]
```

Note that `prore32.exe` is the name of the Report Engine executable. The *db-connection* argument specifies the name of the database containing the Report Engine table and other connection information, such as host and server names. The *parameters* specify any Progress Atlas database connection or startup parameters, or Report Engine startup parameters.

For example, you might use the following statement to invoke Report Engine:

```
OS-COMMAND NO-WAIT prore32.exe -db sample.db -1 -rbtable Rep-Eng -rbdel
```

In this statement, “sample.db” is the name of the database that contains the Rep-Eng table. The -rbtable parameter specifies that Rep-Eng is the Report Engine table, and -rbdel indicates to delete the Report Engine table record for each report when it completes processing the reports.

### Using a DOS window

You can invoke Report Engine directly from the DOS prompt. To run Report Engine from the DOS prompt in a DOS window, use the following syntax:

**SYNTAX**

```
prore32.exe db-connection [ parameters ]
```

Note that `prore32.exe` is the name of the Report Engine executable. The *db-connection* argument specifies the name of the database containing the Report Engine table and other connection information, such as host and server names. The *parameters* specify any Progress startup or database connection parameters, or Report Engine startup parameters.

For example, you might use the following statement to invoke Report Engine:

```
prore32.exe -db sample.db -1 -rbtable Rep-Eng
```

In this statement, "sample.db" is the name of the database that contains the Rep-Eng table and the -rbtable parameter specifies that Rep-Eng is the Report Engine table.

You can also create a .BAT file that uses the utility to run the command line.

### Using an MS-Windows Icon

You can invoke Report Engine from an MS-Windows icon if you enter the command line in the icon's Command Line field. Use the following syntax in the Command Line field:

**SYNTAX**

```
prore32.exe db-connection [ parameters ]
```

Note that `prore32.exe` is the name of the Report Engine. The *db-connection* argument specifies the name of the database containing the Report Engine table and other connection information, such as host and server name. The *parameters* specify any Progress startup or database connection parameters or Report Engine startup parameters.

For example, you might use the following command to invoke Report Engine:

```
prore32.exe -db sample.db -1 -rbtable Rep-Eng -rbupds
```

In this statement, “sample.db” is the name of the database that contains the Rep-Eng table. The -rbtable parameter specifies that Rep-Eng is the Report Engine table, and -rbupds tells Report Engine to update the RB-STATUS field in the Rep-Eng table for each report.

### Using the Start Menu Run Option

You can invoke Report Engine from the Run option from the Start menu. To do so, simply choose Start→ Run, enter the command line in the Command Line field, then choose OK. Use the following syntax in the Command Line field:

#### SYNTAX

```
prore32.exe db-connection [ parameters ]
```

Note that `prore32.exe` is the name of the Report Engine executable. The *db-connection* argument specifies the name of the database containing the Report Engine table and other connection information, such as host and server names. The *parameters* specify any Progress startup or database connection parameters or Report Engine startup parameters.

For example, you might use the following command to invoke Report Engine:

```
prore32.exe -db sample.db -1 -rbtable Rep-Eng
```

In this statement, “sample.db” is the name of the database that contains the Rep-Eng table, and the -rbtable parameter specifies that Rep-Eng is the Report Engine table.

## 4.6 Running Multiple Reports

The table interface allows you to run multiple reports or batches of reports with a single invocation of Report Engine. The advantage to processing multiple reports in this manner is that Report Engine only gets invoked once, thus eliminating the time it takes to start and initialize Report Engine for the second and subsequent reports.

By default, Report Engine processes all the records in the Report Engine table you specify when you invoke it. Therefore, to run multiple reports, simply include all the reports you want to run in a single Report Engine table. To run only a portion of the reports in the Report Engine table, use the Report Tag (-rbtag) startup parameter to pass a value to be matched against the RB-TAG field. For information about the RB-TAG parameter, see Chapter 3, "Report Engine Parameters."

When Report Engine processes multiple reports and encounters a problem or error with one report, it stops processing that report and moves on to the next one. Report Engine stops processing all reports only if it encounters a severe problem, such as the inability to locate a required DLL file or when the Report Engine table is missing a field.

## 4.7 Running Reports from an Application

This section describes how to run reports from 4GL applications. It contains the following examples:

- Running report with a simple filter override (`rbfilt1.p`)
- Prompting for user input for the printer information (`rbprint1.p`)
- Prompting for user input for the filter information (`rbfilt2.p`)
- Generating a list of reports, printing the reports, and updating the RB-STATUS field (`rbplist1.p`)
- Generating a list of reports, prompting the user to select which reports to run, and running the reports (`rblist2.p`)

All the samples illustrate how to run Report Engine with a multi-user database. The samples use the `rbsample.prl` report library and the Progress Sports database.

Follow these steps to run the sample reports:

1 ♦ Create a database called Runtable containing the Report Engine table. You can use the database definition file (`%DLC%\bin\rbreport.df`) that contains loadable definitions for the RBREPORT table. (Note that you can give the database any name you choose, and it can contain other tables, such as the tables that contain your report data.)

2 ♦ Create an MS-Windows icon with the following command on the command line:

```
-mproserv -db dbname -S servername -H hostname -N networktype
```

Note that *dbname* contains the full pathname of the Runtable database.

3 ♦ Double-click the MS-Windows icon to start the database server. The icon becomes minimized.

4 ♦ Start Progress and connect to the Runtable database in multi-user mode using the parameters shown in step 2.

5 ♦ Start the Procedure Editor.

6 ♦ Run the procedures.

To run the procedures, you must specify the appropriate pathname for the Report Engine database and be connected to the Report Engine database in multi-user mode. In the examples, "Runtable" is the Report Engine database name.

### 4.7.1 Running a Report with a Simple Filter Override

The `rbfilt1.p` sample procedure runs the Customer Discount report in the `rbsample.prl` report library with the specified filter override:

**rbfilt1.p**

```
  DO TRANSACTION:

/* 1 */
  CREATE RBREPORT.

/* 2 */
  ASSIGN
    RBREPORT.RB-REPORT-LIBRARY = "c:\dlc\src\aderb\rbsample.prl"
    RBREPORT.RB-REPORT-NAME = "Customer Discount"
    RBREPORT.RB-PRINT-DESTINATION = "D"
    RBREPORT.RB-INCLUDE-RECORDS = "O"
    RBREPORT.RB-FILTER = "IN-RANGE(Order-Line.Discount, 15, 30)"
    RBREPORT.RB-DISPLAY-STATUS = yes
    RBREPORT.RB-DISPLAY-ERRORS = yes.

/* 3 */
  RELEASE RBREPORT.

  END.

/* 4 */
  RUN aderb\_prore.p(false,
    "-db Runtable -S servername -H hostname -N networktype -rbdel").
```

The commented numbers correspond to the following step-by-step descriptions:

1. Create the RBREPORT record.
2. Assign values to the record, including the filter override condition.
3. Explicitly release the RBREPORT record so that Report Engine can use it.
4. Call _prore to invoke Report Engine and run the report with the filter override. Then use the Report Delete (-rbdel) parameter to delete the record from the Report Engine table when the procedure is done.

### 4.7.2 Prompting for User Input for the Printer Information

The rbprint1.p sample procedure runs the Accts Receivable report in the rbsample.prl report library and prompts the user for the printer destination and printer name using the Report Engine built-in prompting mechanism:

**rbprint1.p**

```
  DO TRANSACTION:

/* 1 */
  CREATE RBREPORT.

/* 3 */
  ASSIGN
    RBREPORT.RB-REPORT-LIBRARY = "c:\dlc\src\aderb\rbsample.prl"
    RBREPORT.RB-REPORT-NAME = "Accts Receivable"
    RBREPORT.RB-PRINT-DESTINATION = "?"
    RBREPORT.RB-PRINTER-NAME = "?"
    RBREPORT.RB-DISPLAY-STATUS = yes
    RBREPORT.RB-DISPLAY-ERRORS = yes.

/* 3 */
  RELEASE RBREPORT.

  END.

/* 4 */
  RUN aderb\_prore.p(false,
    "-db Runtable -S servername -H hostname -N networktype -rbdel").
```

The commented numbers correspond to the following step-by-step descriptions:

1. Create the RBREPORT record.

2. Assign values to the record, but specify questions marks (?) for the printer destination and printer name fields. The question marks instruct Report Engine to prompt the user for input for those fields.

3. Explicitly release the RBREPORT record so that Report Engine can use it.

4. Call _prore to invoke Report Engine and run the report with the user prompts. As shown in the previous sample procedure, you use the Report Delete (-rbdel) parameter to delete the record from the Report Engine table when the procedure is done.

### 4.7.3 Prompting for User Input for the Filter Information

The `rbfilt2.p` sample procedure runs the Customer List report in the `rbsample.prl` report library and prompts the user for minimum and maximum values for the filter condition:

**rbfilt2.p**

```
  DEF VAR high-value AS INTEGER INITIAL 0.
  DEF VAR low-value AS INTEGER INITIAL 0.
  DEF VAR rb-filter-value AS CHARACTER INITIAL "".

/* 1 */
  FORM" Enter Low  Value for CUSTOMER NUMBER: " low-value at 20  SKIP
    "Enter High Value for CUSTOMER NUMBER: " high-value at 20
    WITH FRAME TEST-FRAME CENTERED NO-LABELS.

/* 2 */
  UPDATE low-value high-value WITH FRAME TEST-FRAME.
  HIDE FRAME TEST-FRAME.

/* 3 */
  rb-filter-value = "Customer.Cust-num >= " + STRING(low-value) +
                    " AND Customer.Cust-num <= " + STRING(high-value).

/* 4 */
  DO TRANSACTION:

    CREATE RBREPORT.

    ASSIGN
      RBREPORT.RB-REPORT-LIBRARY = "c:\dlc\src\aderb\rbsample.prl"
      RBREPORT.RB-REPORT-NAME = "Customer List"
      RBREPORT.RB-PRINT-DESTINATION = "D"
      RBREPORT.RB-INCLUDE-RECORDS  = "O"
      RBREPORT.RB-FILTER  = rb-filter-value
      RBREPORT.RB-DISPLAY-ERRORS = yes
      RBREPORT.RB-DISPLAY-STATUS = yes.

      RELEASE RBREPORT.

  END.

  RUN  aderb\_prore.p(false,
    "-db Runtable  -S servername -H hostname -N networktype -rbdel").
```

The commented numbers correspond to the following step-by-step descriptions:

1. Define a form for the prompt.
2. Prompt the user for filter information.
3. Assign the filter override condition to rb-filter-value.
4. Run the Customer List report with the filter override on the report.

### 4.7.4 Generating a List of Reports, Printing the Reports, and Updating the RB-STATUS Field

The `rbplist1.p` sample procedure runs all the reports in the `rbsample.prl` report library and updates the RB-STATUS field. This procedure assumes that all of the reports in the list generated by `_getname.p` use the Sports database. Report Engine connects to the Sports database only once on startup, and the same connection is used for each report since the '=>' syntax is used in the RB-DB-CONNECTION field of the RBREPORT table.

To run the `rbplist1.p` procedure with the `rbsample.prl` report library, execute the following command:

```
RUN rblist1.p("c:\dlc\src\aderb\rbsample.prl").
```

Here is the rbplist1.p procedure:

**rbplist1.p**

```
/*1/
  DEF VAR report-list AS CHARACTER INITIAL "".
  DEF VAR report-count AS INTEGER INITIAL 0.
  DEF VAR report-number AS INTEGER INITIAL 0.
  DEF VAR current-report-name AS CHARACTER INITIAL "".

/* 2 */
  DEF INPUT PARAMETER report-library AS CHARACTER.

/* 3 */
  RUN aderb/_getname.p(report-library, OUTPUT report-list,
    OUTPUT report-count).

/* 4 */
  REPEAT report-number = 1 TO report-count:

    CREATE RBreport.

    current-report-name = ENTRY(report-number, report-list).

    ASSIGN
      RBREPORT.RB-REPORT-LIBRARY = report-library
      RBREPORT.RB-REPORT-NAME = current-report-name
      RBREPORT.RB-DB-CONNECTION = "sports => logsport"
      RBREPORT.RB-PRINT-DESTINATION = "D"
      RBREPORT.RB-DISPLAY-STATUS = yes
      RBREPORT.RB-DISPLAY-ERRORS = yes.

    RELEASE RBREPORT.

  END.

/* 5 */
  RUN aderb\_prore.p(false,"-db Runtable -S servername -H hostname
    -N networktype -db c:\data\sports -1 -ld logsport -rbupds").

/* 6 */
  FOR EACH RBREPORT:

    MESSAGE "Status of Report '" + RBREPORT.RB-REPORT-NAME +
      "': " + RBREPORT.RB-STATUS VIEW-AS ALERT-BOX.

    DELETE RBREPORT.

  END.
```

The commented numbers correspond to the following step-by-step descriptions:

1. Define the variables used in the procedure.
2. Define the input parameter to the procedure.
3. Run `_getname.p` to generate a list of reports in the specified library.
4. Add the reports to the Runtable database.
5. Run each report record in the RBREPORT table using the connection to the database on the command line. Then, update the RB-STATUS field.
6. For each record, view RB-STATUS field as an Alert Box, then delete the record.

### 4.7.5 Generating a List of Reports, Prompting the User to Select Which Reports to Run, and Running the Reports

The `rblist2.p` sample procedure generates a list of all the reports in the `rbsample.prl` report library, prompts the user to specify which reports to run, then runs the reports. This procedure assumes that all of the reports in the list generated by `_getname.p` use the Sports database. Report Engine connects to the database only once on startup, and the same connection is used for each report since the '=>' syntax is used in the RB-DB-CONNECTION field of the RBREPORT table.

To run the `rblist2.p` procedure with the `rbsample.prl` report library, execute the following command:

```
RUN rblist2.p("c:\dlc\src\aderb\rbsample.prl").
```

Here is the rbplist2.p procedure:

**rbplist2.p**

```
/* 1 */
  DEF VAR report-list AS CHARACTER INITIAL "".
  DEF VAR report-count AS INTEGER INITIAL 0.
  DEF VAR report-number AS INTEGER INITIAL  0.
  DEF VAR current-report-name AS CHARACTER INITIAL "".

/* 2 */
  DEF INPUT PARAMETER report-library AS CHARACTER.

/* 3 */
  DEF VAR user-answer AS LOGICAL.

/* 4 */
  RUN aderb/_getname.p(report-library, OUTPUT report-list,
    OUTPUT report-count).

/* 5 */
  DO report-number = 1 TO report-count:

    current-report-name = ENTRY(report-number, report-list).

    MESSAGE  "Report Number" +  STRING(report-number) + " is '"
      + current-report-name + "'.  Do you want to print it?"
      VIEW-AS  ALERT-BOX  QUESTION  BUTTONS  yes-no
      UPDATE  user-answer.

    IF (user-answer) THEN DO TRANSACTION:

      CREATE RBREPORT.

      ASSIGN
        RBREPORT.RB-REPORT-LIBRARY = report-library
        RBREPORT.RB-REPORT-NAME = current-report-name
        RBREPORT.RB-DB-CONNECTION = "sports => logsport"
        RBREPORT.RB-PRINT-DESTINATION = "D"
        RBREPORT.RB-DISPLAY-STATUS = yes
        RBREPORT.RB-DISPLAY-ERRORS = yes.

      RELEASE RBREPORT.

    END.
  END.

/* 6 */
  RUN aderb\_prore.p(false,"-db Runtable -S servername -H hostname
    -N networktype -db c:\data\sports -1 -ld logsport -rbdel").
```

The commented numbers correspond to the following step-by-step descriptions:

1. Define the variables used in the procedure.
2. Define the input parameter to the procedure.
3. Define the user-answer variable to handle the user's input.
4. Run `_getname.p` to generate a list of reports in the specified library.
5. Prompt the user and add the selected reports to the Runtable database.
6. Run the selected reports, then delete the table records.

# 5

# Report Engine PRINTRB and PRNTRB2 Interfaces

The Report Engine PRINTRB and PRNTRB2 interfaces allow you to invoke Report Engine from the Progress 4GL using parameters instead of tables and fields to specify report parameters.

This chapter describes:

- PRINTRB/PRNTRB2 interface basics
- PRINTRB interface quick start example
- PRNTRB2 interface quick start example
- Using PRINTRB/PRNTRB2 parameters
- Running reports from an application

## 5.1 PRINTRB/PRNTRB2 Interface Basics

The PRINTRB and PRNTRB2 interfaces are Progress 4GL procedures that allow you to control the report processing by specifying report parameters. Because they are Progress procedures, you can run them only from a 4GL application. This means that you must be in a Progress session to run the Report Engine PRINTRB or PRNTRB2 interface.

Follow these general steps to use the PRINTRB/PRNTRB2 interface:

1. Start Progress.
2. For PRINTRB, specify each of the nineteen parameters for the report in the RUN statement that calls the `aderb/_printrb.p` procedure file.
3. For PRNTRB2, specify each of the twenty parameters for the report in the RUN statement that calls the `aderb/_prntrb2.p` procedure file.

**NOTE:** The PRINTRB interface does not support all Report Builder Version 8.0 functionality and is provided solely for backward compatibility.

## 5.2 PRINTRB Interface Quick Start Example

There are two steps you must perform to use this interface. The first step in using the PRINTRB interface is to start Progress. The second step is to specify all the parameter values for the report. To show you how to perform the second task, Progress Software provides the `rbstart2.p` procedure that runs the `aderb/_printrb.p` procedure file and specifies all the parameter values.

Follow these steps to run the PRINTRB quick start example:

1 ♦ Access the Progress Procedure Editor.

2 ♦ Run the `rbstart2.p` procedure.

Here is the code for the rbstart2.p procedure:

**rbstart2.p**

```
RUN aderb\_printrb (
  "c:\dlc\src\aderb\rbsample.prl",  /* RB-REPORT-LIBRARY */
  "Customer List",                  /* RB-REPORT-NAME */
  "",                               /* RB-DB-CONNECTION */
  "",                               /* RB-INCLUDE-RECORDS */
  "",                               /* RB-FILTER */
  "",                               /* RB-MEMO-FILE */
  "D",                              /* RB-PRINT-DESTINATION */
  "",                               /* RB-PRINTER-NAME */
  "",                               /* RB-PRINTER-PORT */
  "",                               /* RB-OUTPUT-FILE */
  0,                                /* RB-NUMBER-COPIES  - zero */
  0,                                /* RB-BEGIN-PAGE - zero */
  0,                                /* RB-END-PAGE - zero */
  no,                               /* RB-TEST-PATTERN */
  "",                               /* RB-WINDOW-TITLE */
  yes,                              /* RB-DISPLAY-ERRORS */
  yes,                              /* RB-DISPLAY-STATUS */
  no,                               /* RB-NO-WAIT */
  "").                              /* RB-OTHER-PARAMETERS */
```

## 5.3 PRNTRB2 Interface Example

There are two steps you must perform to use this interface. The first step in using the PRNTRB2 interface is to start Progress. The second step is to specify all the parameter values for the report. To show you how to perform the second task, Progress Software provides the rbstart3.p procedure that runs the aderb/_printrb.p procedure file and specifies all the parameter values.

Follow these steps to run the PRINTRB quick start example:

1 ♦ Access the Progress Procedure Editor.

2 ♦ Run the rbstart3.p procedure.

Here is the code for the rbstart3.p procedure:

**rbstart3.p**

```
RUN aderb\_printrb(
  "c:\dlc\src\aderb\rbsample.prl",  /* RB-REPORT-LIBRARY */
  "Customer List",                 /* RB-REPORT-NAME */
  "",                              /* RB-DB-CONNECTION */
  "",                              /* RB-INCLUDE-RECORDS */
  "",                              /* RB-FILTER */
  "",                              /* RB-MEMO-FILE */
  "D",                             /* RB-PRINT-DESTINATION */
  "",                              /* RB-PRINTER-NAME */
  "",                              /* RB-PRINTER-PORT */
  "",                              /* RB-OUTPUT-FILE */
  0,                               /* RB-NUMBER-COPIES  - zero */
  0,                               /* RB-BEGIN-PAGE - zero */
  0,                               /* RB-END-PAGE - zero */
  no,                              /* RB-TEST-PATTERN */
  "",                              /* RB-WINDOW-TITLE */
  yes,                             /* RB-DISPLAY-ERRORS */
  yes,                             /* RB-DISPLAY-STATUS */
  no,                              /* RB-NO-WAIT */
  "",                              /* RB-OTHER-PARAMETERS */
  "status.out").                   /* RB-STATUS-FILE */
```

## 5.4 Using the PRINTRB/PRNTRB2 Parameters

There are predefined parameters that you must specify when using the PRINTRB or PRNTRB2 interface. These parameters identify the report you want to run and control frequently changed report features, such as filters and print information, allowing you to customize your reports. You specify these parameters as arguments to a procedure, then run the procedure from a Progress 4GL program.

When you use the PRINTRB or PRNTRB2 interface, you must enter all the parameter values in the order in which the parameters appear in Table 5–1 regardless of whether you want to override the default value.

Table 5–1 lists the predefined Report Engine parameters.

**Table 5–1: Predefined Report Engine Parameters** *(1 of 2)*

| Parameter Name | Contents | Data Type |
|---|---|---|
| RB-REPORT-LIBRARY | Report library name | CHARACTER |
| RB-REPORT-NAME | Report name | CHARACTER |
| RB-DB-CONNECTION | Database connection override string | CHARACTER |
| RB-INCLUDE-RECORDS | Filter flag | CHARACTER |
| RB-FILTER | Filter expression | CHARACTER |
| RB-MEMO-FILE | Memo filename | CHARACTER |
| RB-PRINT-DESTINATION | Print destination | CHARACTER |
| RB-PRINTER-NAME | Printer name | CHARACTER |
| RB-PRINTER-PORT | Printer port | CHARACTER |
| RB-OUTPUT-FILE | Output filename | CHARACTER |
| RB-NUMBER-COPIES | Number of copies | INTEGER |
| RB-BEGIN-PAGE | Beginning page number | INTEGER |
| RB-END-PAGE | Ending page number | INTEGER |
| RB-TEST-PATTERN | Test pattern flag | LOGICAL |
| RB-WINDOW-TITLE | Window title | CHARACTER |
| RB-DISPLAY-ERRORS | Display error flag | LOGICAL |
| RB-DISPLAY-STATUS | Display status flag | LOGICAL |
| RB-NO-WAIT | Synchronous or asynchronous processing | LOGICAL |

**Table 5–1: Predefined Report Engine Parameters** *(2 of 2)*

| Parameter Name | Contents | Data Type |
|---|---|---|
| RB-OTHER-PARAMETERS | User-defined parameters | N/A |
| RB-STATUS-FILE | Status filename<br>Required for PRNTRB2 only | CHARACTER |

See Chapter 3, "Report Engine Parameters," for a detailed description of each parameter.

## 5.5 Running Report Engine from an Application

This section describes how to run reports from Progress 4GL applications. It contains the following examples:

- Running report with a simple filter override (`rbfilt3.p`)
- Prompting for user input for the printer information (`rbprint2.p`)
- Prompting for user input for the filter information (`rbfilt4.p`)
- Running multiple reports in a batch (`rbbatch.p`)
- Running multiple reports and prompting for user input (`rbprompt.p`)
- Overriding database connection information (`rbdbconn.p`)

All the samples illustrate how to run Report Engine with a local multi-user database. The samples use the `rbsample.prl` report library and the Progress Sports database. The `rbsample.prl` report library is located in `%DLC%\SRC\aderb`.

You can run reports from a 4GL application by running the `_printrb.p` procedure. This procedure runs the report using the parameters you specify. You specify the desired parameters for each report in the RUN statement when you run Report Engine.

Follow these steps to run a report with Report Engine:

1 ♦ Create the report definition in Report Builder.

2 ♦ Use the following command in the Progress 4GL application:

```
RUN aderb/_printrb.p (parameters)
```

**NOTE:** *Parameters* are described in Chapter 3, "Report Engine Parameters."

Remember that the startup parameters in PROSTARTUP and RBSTARTUP files apply to the PRINTRB and PRNTRB2 interfaces even though there is no explicit command line available.

### 5.5.1 Running a Report with a Simple Filter Override

The `rbfilt3.p` sample procedure runs the Customer Discount report in the `rbsample.prl` report library with the specified filter override:

**rbfilt3.p**

```
RUN aderb\_printrb(
  "c:\dlc\src\aderb\rbsample.prl",   /* RB-REPORT-LIBRARY */
  "Customer Discount",               /* RB-REPORT-NAME */
  "",                                /* RB-DB-CONNECTION */
  "O",/* RB-INCLUDE-RECORDS - letter O */
  "IN-LIST(Order-Line.Discount,
  10,15,35) > 0",                /* RB-FILTER */
  "",                                /* RB-MEMO-FILE */
  "D",                               /* RB-PRINT-DESTINATION */
  "",                                /* RB-PRINTER-NAME */
  "",                                /* RB-PRINTER-PORT */
  "",                                /* RB-OUTPUT-FILE */
  0,                                 /* RB-NUMBER-COPIES - zero */
  0,                                 /* RB-BEGIN-PAGE - zero */
  0,                                 /* RB-END-PAGE - zero */
  no,                                /* RB-TEST-PATTERN */
  "",                                /* RB-WINDOW-TITLE */
  yes,                               /* RB-DISPLAY-ERRORS */
  yes,                               /* RB-DISPLAY-STATUS */
  no,                                /* RB-NO-WAIT */
  "").                               /* RB-OTHER-PARAMETERS */
```

Notice that the procedure specifies all nineteen of the PRINTRB interface parameters in the order in which they appear, even though it overrides only the filter condition values.

**NOTE:** If you are using the PRNTRB2 interface you must modify the procedure to run aderb_prntrb2 and include the RB-STATUS-FILE parameter.

### 5.5.2 Prompting for User Input for the Printer Information

The `rbprint2.p` sample procedure runs the Accts Receivable report in the `rbsample.prl` report library and prompts the user for the printer destination and printer name using the Report Engine built-in prompting mechanism:

**rbprint2.p**

```
RUN aderb\_printrb(
  "c:\dlc\src\aderb\rbsample.prl",  /* RB-REPORT-LIBRARY */
  "Accts Receivable",              /* RB-REPORT-NAME */
  "",                              /* RB-DB-CONNECTION */
  "",                              /* RB-INCLUDE-RECORDS */
  "",                              /* RB-FILTER */
  "",                              /* RB-MEMO-FILE */
  "?",                             /* RB-PRINT-DESTINATION */
  "?",                             /* RB-PRINTER-NAME */
  "",                              /* RB-PRINTER-PORT */
  "",                              /* RB-OUTPUT-FILE */
  0,                               /* RB-NUMBER-COPIES - zero */
  0,                               /* RB-BEGIN-PAGE - zero */
  0,                               /* RB-END-PAGE - zero */
  no,                              /* RB-TEST-PATTERN */
  "",                              /* RB-WINDOW-TITLE */
  yes,                             /* RB-DISPLAY-ERRORS */
  yes,                             /* RB-DISPLAY-STATUS */
  no,                              /* RB-NO-WAIT */
  "").                             /* RB-OTHER-PARAMETERS */
```

Although you might expect to see code that explicitly prompts the user for the printer destination and printer port, you do not have to do this. Simply enter question marks (?) in the RB-PRINTER-NAME and RB-PRINTER-PORT parameters, and Report Engine automatically allows the user to select the print destination (screen or printer) and printer at run time.

**NOTE:** If you are using the PRNTRB2 interface you must modify the procedure to run aderb_prntrb2 and include the RB-STATUS-FILE parameter.

### 5.5.3 Prompting for User Input for the Filter Information

The `rbfilt4.p` sample procedure runs the Customer List report in the `rbsample.prl` report library and prompts the user for minimum and maximum values for the filter condition:

**rbfilt4.p**

```
  DEF VAR high-value AS INTEGER INITIAL 0.
  DEF VAR low-value AS INTEGER INITIAL 0.
  DEF VAR rb-filter-value  AS CHARACTER INITIAL "".

/* 1 */
  FORM  "Enter Low  Value for CUSTOMER NUMBER: "  low-value at 20  SKIP
    "Enter High Value for CUSTOMER NUMBER: " high-value at 20
    WITH FRAME TEST-FRAME CENTERED NO-LABELS.

/* 2 */
  UPDATE low-value high-value WITH FRAME TEST-FRAME.
  HIDE FRAME TEST-FRAME.

/* 3 */
  rb-filter-value = "Customer.Cust-num >= " + STRING(low-value) +
    " AND Customer.Cust-num <= " + STRING(high-value).

/* 4 */
  RUN aderb\_printrb(
    "c:\dlc\src\aderb\rbsample.prl", /* RB-REPORT-LIBRARY */
    "Customer List",                 /* RB-REPORT-NAME */
    "",                              /* RB-DB-CONNECTION */
    "O",                             /* RB-INCLUDE-RECORDS - letter O*/
    rb-filter-value,                 /* RB-FILTER */
    "",                              /* RB-MEMO-FILE */
    "D",                             /* RB-PRINT-DESTINATION */
    "",                              /* RB-PRINTER-NAME */
    "",                              /* RB-PRINTER-PORT */
    "",                              /* RB-OUTPUT-FILE */
    0,                               /* RB-NUMBER-COPIES - zero */
    0,                               /* RB-BEGIN-PAGE -zero */
    0,                               /* RB-END-PAGE - zero */
    no,                              /* RB-TEST-PATTERN */
    "",                              /* RB-WINDOW-TITLE */
    yes,                             /* RB-DISPLAY-ERRORS */
    yes,                             /* RB-DISPLAY-STATUS */
    no,                              /* RB-NO-WAIT */
    "")                              /* RB-OTHER-PARAMETERS */
```

The commented numbers correspond to the following step-by-step descriptions:

1. Define a form for the prompt.
2. Prompt the user for filter information.
3. Assign the filter override condition to rb-filter-value.
4. Run the Customer List report with the filter override.

**NOTE:** If you are using the PRNTRB2 interface you must modify the procedure to run aderb_prntrb2 and include the RB-STATUS-FILE parameter.

### 5.5.4 Running Multiple Reports in a Batch

You can run all the reports in the `rbsample.prl` report library as a batch job with the same set of parameters using `rbbatch.p`.

To run the `rbbatch.p` procedure with the `rbsample.prl` report library, execute the following command:

```
RUN rbbatch.p ("c:\dlc\src\aderb\rbsample.prl")
```

Here is the rbbatch.p procedure:

**rbbatch.p** *(1 of 2)*

```
/* 1 */
  DEF VAR RB-REPORT-NAME AS CHARACTER INITIAL "".
  DEF VAR RB-DB-CONNECTION AS CHARACTER INITIAL "".
  DEF VAR RB-INCLUDE-RECORDS AS CHARACTER INITIAL "".
  DEF VAR RB-FILTER AS CHARACTER INITIAL "".
  DEF VAR RB-MEMO-FILE AS CHARACTER INITIAL "".
  DEF VAR RB-PRINT-DESTINATION AS CHARACTER INITIAL "".
  DEF VAR RB-PRINTER-NAME AS CHARACTER INITIAL "".
  DEF VAR RB-PRINTER-PORT AS CHARACTER INITIAL "".
  DEF VAR RB-OUTPUT-FILE AS CHARACTER INITIAL "".
  DEF VAR RB-NUMBER-COPIES AS INTEGER INITIAL 1.
  DEF VAR RB-BEGIN-PAGE AS INTEGER INITIAL 0.
  DEF VAR RB-END-PAGE AS INTEGER INITIAL 0.
  DEF VAR RB-TEST-PATTERN AS LOGICAL INITIAL no.
  DEF VAR RB-WINDOW-TITLE AS CHARACTER INITIAL "".
  DEF VAR RB-DISPLAY-ERRORS AS LOGICAL INITIAL yes.
  DEF VAR RB-DISPLAY-STATUS AS LOGICAL INITIAL yes.
  DEF VAR RB-NO-WAIT AS LOGICAL INITIAL no.
  DEF VAR RB-OTHER-PARAMETERS AS CHARACTER INITIAL "".

/* 2 */
  DEF VAR report-list AS CHARACTER.
  DEF VAR report-count AS INTEGER.
  DEF VAR report-number AS INTEGER.

/* 3 */
  DEF INPUT PARAMETER report-library AS CHARACTER.

/* 4 */
  RUN aderb/_getname.p (report-library, OUTPUT report-list,
    OUTPUT report-count).
```

**rbbatch.p** *(2 of 2)*

```
/* 5 */
  DO report-number = 1 to report-count:

    RB-REPORT-NAME = entry (report-number, report-list).

    RUN aderb/_printrb (report-library,
      RB-REPORT-NAME,
      RB-DB-CONNECTION,
      RB-INCLUDE-RECORDS,
      RB-FILTER,
      RB-MEMO-FILE,
      RB-PRINT-DESTINATION,
      RB-PRINTER-NAME,
      RB-PRINTER-PORT,
      RB-OUTPUT-FILE,
      RB-NUMBER-COPIES,
      RB-BEGIN-PAGE,
      RB-END-PAGE,
      RB-TEST-PATTERN,
      RB-WINDOW-TITLE,
      RB-DISPLAY-ERRORS,
      RB-DISPLAY-STATUS,
      RB-NO-WAIT,
      RB-OTHER-PARAMETERS).

  END.
```

The commented numbers correspond to the following step-by-step descriptions:

1. Define variables for the report parameters. Note that you must specify each parameter for each report that you run, but that you can assign each parameter a default value for this procedure.

2. Define report-list and report-count for the `_getname.p` output, and report-number to keep track of the reports.

3. Define input parameter for the library name.

4. Run `_getname.p` to access all the reports in the `rbsample.prl` library.

5. Print all the reports in the library.

**NOTE:** If you are using the PRNTRB2 interface you must modify the procedure to run aderb_prntrb2 and include the RB-STATUS-FILE parameter.

### 5.5.5 Running Multiple Reports and Prompting for User Input

In the following example, `rbprompt.p`, you prompt the user to specify whether to generate each report in the `rbsample.prl` report library.

To run the `rbprompt.p` procedure with the `rbsample.prl` report library, execute the following command:

```
RUN rbprompt.p ("c:\dlc\src\aderb\rbsample.prl")
```

Here is the `rbprompt.p` procedure:

**rbprompt.p** *(1 of 2)*

```
/* 1 */
  DEF VAR RB-REPORT-NAME AS CHARACTER INITIAL "".
  DEF VAR RB-DB-CONNECTION AS CHARACTER INITIAL "".
  DEF VAR RB-INCLUDE-RECORDS AS CHARACTER INITIAL "".
  DEF VAR RB-FILTER AS CHARACTER INITIAL "".
  DEF VAR RB-MEMO-FILE AS CHARACTER INITIAL "".
  DEF VAR RB-PRINT-DESTINATION AS CHARACTER INITIAL "".
  DEF VAR RB-PRINTER-NAME AS CHARACTER INITIAL "".
  DEF VAR RB-PRINTER-PORT AS CHARACTER INITIAL "".
  DEF VAR RB-OUTPUT-FILE AS CHARACTER INITIAL "".
  DEF VAR RB-NUMBER-COPIES AS INTEGER INITIAL 1.
  DEF VAR RB-BEGIN-PAGE AS INTEGER INITIAL 0.
  DEF VAR RB-END-PAGE AS INTEGER INITIAL 0.
  DEF VAR RB-TEST-PATTERN AS LOGICAL INITIAL no.
  DEF VAR RB-WINDOW-TITLE AS CHARACTER INITIAL "".
  DEF VAR RB-DISPLAY-ERRORS AS LOGICAL INITIAL yes.
  DEF VAR RB-DISPLAY-STATUS AS LOGICAL INITIAL yes.
  DEF VAR RB-NO-WAIT AS LOGICAL INITIAL no.
  DEF VAR RB-OTHER-PARAMETERS AS CHARACTER INITIAL "".

  DEF VAR report-list AS CHARACTER.
  DEF VAR report-count AS INTEGER.
  DEF VAR report-number AS INTEGER.

/* 2 */
  DEF INPUT PARAMETER report-library AS CHARACTER.

/* 3 */
  DEF VAR user-answer AS LOGICAL.
```

**rbprompt.p** *(2 of 2)*

```
/* 4 */
  RUN aderb/_getname.p (report-library, OUTPUT report-list,
    OUTPUT report-count).

/* 5 */
  DO report-number = 1 to report-count:

    RB-REPORT-NAME = ENTRY (report-number, report-list).

    MESSAGE "Report Number " + string (report-number) + " is "
      + RB-REPORT-NAME + ". Do you want to print it?"
      VIEW-AS ALERT-BOX QUESTION BUTTONS yes-no UPDATE user-answer.

    IF (user-answer) THEN
      RUN aderb/_printrb.p (report-library,
        RB-REPORT-NAME,
        RB-DB-CONNECTION,
        RB-INCLUDE-RECORDS,
        RB-FILTER,
        RB-MEMO-FILE,
        RB-PRINT-DESTINATION,
        RB-PRINTER-NAME,
        RB-PRINTER-PORT,
        RB-OUTPUT-FILE,
        RB-NUMBER-COPIES,
        RB-BEGIN-PAGE,
        RB-END-PAGE,
        RB-TEST-PATTERN,
        RB-WINDOW-TITLE,
        RB-DISPLAY-ERRORS,
        RB-DISPLAY-STATUS,
        RB-NO-WAIT,
        RB-OTHER-PARAMETERS).
    END.
```

The commented numbers correspond to the following step-by-step descriptions:

1. Define variables for the report parameters.

2. Define the input parameter for the library name.

3. Define the user-answer variable to handle the user's input.

4. Run _getname.p to generate a list of reports in the rbsample.prl library.

5. Prompt the user to specify which reports to print, and print the reports.

**NOTE:** If you are using the PRNTRB2 interface you must modify the procedure to run aderb_prntrb2 and include the RB-STATUS-FILE parameter.

### 5.5.6 Overriding Database Connection Information

The following example, rbdbconn.p, illustrates how to override the database connection for a report. If the report was saved with a local database in single-user mode and you want to override it with a remote database in multi-user mode, run the rbdbconn.p procedure.

Here is the rbdbconn.p procedure:

**rbdbconn.p**

```
RUN aderb\_printrb(
  "c:\dlc\src\aderb\rbsample.prl", /* RB-REPORT-LIBRARY */
  "Accts Receivable",              /* RB-REPORT-NAME */
  "sports = -db sports -H newhost
  -S newserver -N TCP",         /* RB-DB-CONNECTION */
  "",                              /* RB-INCLUDE-RECORDS */
  "",                              /* RB-FILTER */
  "",                              /* RB-MEMO-FILE */
  "",                              /* RB-PRINT-DESTINATION */
  "",                              /* RB-PRINTER-NAME */
  "",                              /* RB-PRINTER-PORT */
  "",                              /* RB-OUTPUT-FILE */
  0,                               /* RB-NUMBER-COPIES - zero */
  0,                               /* RB-BEGIN-PAGE - zero */
  0,                               /* RB-END-PAGE - zero */
  no,                              /* RB-TEST-PATTERN */
  "",                              /* RB-WINDOW-TITLE */
  yes,                             /* RB-DISPLAY-ERRORS */
  yes,                             /* RB-DISPLAY-STATUS */
  no,                              /* RB-NO-WAIT */
  "").                             /* RB-OTHER-PARAMETERS */
```

**NOTE:** If you are using the PRNTRB2 interface you must modify the procedure to run aderb_prntrb2 and include the RB-STATUS-FILE parameter.

# Index


## A

Applications
  running reports 5–6

Audience ix

## B

Bold typeface
  as typographical convention x

## D

Database connection, overriding 5–15

Database definition files 4–2

defprint.p procedure 2–4

Drivers
  printer 1–5

## E

Encoded Password (-rbP) startup parameter 2–9, 4–12

Error codes 2–12

Error messages 2–12, 2–13
  displaying descriptions xiv

Errors
  RO-ERROR-CODE 2–12
  RO-ERROR-MESSAGE 2–12

## F

Fields
  RB-STATUS 2–13

Files
  rbreport.df 4–2
  rbrun.out 2–10
  rbstart.pf 4–8
  Report Status 2–10
  startup.pf 4–8
  _prore.p 4–15

Forms
  printing 3–20

## G

Generating reports
  listing 2–2

_getini.p procedure 2–7

_getname.p procedure 2–2, 5–11

## H

Help
xiv

## I

Italic typeface
as typographical convention x

## K

Keystrokes x

## L

Listing reports
_getname.p procedure 5–11

## M

Manual
organization of ix
syntax notation xi

Messages
displaying descriptions xiv

Monospaced typeface
as typographical convention x

Multiple reports 4–19

## O

OS-COMMAND statement 4–16

Overriding a database connection 5–15

## P

Pages 2–13

Parameters
predefined 3–3, 4–4
RUNTIME-PARAMETER 3–28
user defined 3–20

Parameters in report definitions 3–28

Passwords
_rbpwenc.p procedure 2–9

_prdef.p procedure 2–4

Predefined parameters 3–2, 3–26, 4–4

Printer driver considerations 1–5

Printing
test patterns 3–20
using forms 3–20

printlst.p procedure 2–6

PRINTRB interface
_printrb.p procedure 5–2

_printrb.p procedure 5–2

_prlist.p procedure 2–5

Procedures
defprint.p 2–4
_getini.p 2–7
_getname.p 2–2, 5–11
_prdef.p 2–4
printlst.p 2–6
_printrb.p 5–2
_prlist.p 2–5
progini.p 2–8
_rbpwenc.p 2–9
reptlist.p 2–3

progini.p procedure 2–8

Prompting for user input
predefined parameters 3–26
user-defined parameters 3–26

_prore.p procedure 4–15

## R

-rbcpreportin startup paramete 4–9

-rbcpudfin startup parameter 4–10

-rbdel startup parameter 4–10

-rbP startup parameter 2–9, 4–12

_rbpwenc.p procedure 2–9

rbreport.df file 4–2

rbrun.out file 2–10

-rbstatfile startup parameter 2–10, 4–11

-rbtable startup parameter 4–13

-rbtag startup paramete 4–13

-rbupds startup parameter 2–13, 4–14

rbstart.pf file 4–8

RB-BEGIN-PAGE predefined Report Engine parameter 3–3, 3–4, 5–5

RB-BEGIN-PAGE Report Engine table field name 4–5

RB-DB-CONNECTION predefined Report Engine parameter 3–3, 3–5, 5–5

RB-DB-CONNECTION Report Engine table field name 4–4

RB-DISPLAY ERRORS predefined Report Engine parameter 3–3, 5–5

RB-DISPLAY ERRORS Report Engine table field name 4–5

RB-DISPLAY-STATUS predefined Report Engine parameter 3–4, 3–11, 5–5

RB-DISPLAY-STATUS Report Engine table field name 4–5

RB-END-PAGE predefined Report Engine parameter 3–3, 3–11, 5–5

RB-END-PAGE Report Engine table field name 4–5

RB-FILTER predefined Report Engine parameter 3–3, 3–11, 5–5

RB-FILTER Report Engine table field name 4–4

RB-INCLUDE-RECORDS predefined Report Engine parameter 3–3, 3–13, 5–5

RB-INCLUDE-RECORDS Report Engine table field name 4–4

RB-MEMO-FILE predefined Report Engine parameter 3–3, 3–14, 5–5

RB-MEMO-FILE Report Engine table field name 4–4

RB-NO-WAIT predefined Report Engine parameter 3–4, 3–14, 5–5

RB-NUMBER-COPIES predefined Report Engine parameter 3–3, 3–14, 5–5

RB-NUMBER-COPIES Report Engine table field name 4–5

RB-OUTPUT-FILE predefined Report Engine parameter 3–3, 3–15, 3–19, 5–5

RB-OUTPUT-FILE Report Engine table field name 4–5

RB-PRINT-DESTINATION predefined Report Engine parameter 3–3, 3–15, 5–5

RB-PRINT-DESTINATION Report Engine table field name 4–4

RB-PRINTER-NAME predefined Report Engine parameter 3–3, 3–16

RB-PRINTER-NAME Report Engine table field name 4–4

RB-PRINTER-NAME Report predefined Report Engine parameter 5–5

RB-PRINTER-PORT predefined Report Engine parameter 3–3, 3–17, 5–5

RB-PRINTER-PORT Report Engine table field name 4–4

RB-REPORT-LIBRARY predefined Report Engine parameter 3–3, 3–17, 5–5

RB-REPORT-LIBRARY Report Engine table field name 4–4

RB-REPORT-NAME predefined Report Engine parameter 3–3, 3–18, 5–5

RB-REPORT-NAME Report Engine table field name 4–4, 4–5

RB-STATUS field 2–13

RB-STATUS predefined Report Engine parameter 3–18

RB-STATUS Report Engine table field name 4–5

RB-TAG predefined Report Engine parameter 3–19

RB-TEST-PATTERN predefined Report Engine parameter 3–3, 3–20, 5–5

RB-TEST-PATTERN Report Engine table field name 4–5

RB-WINDOW-TITLE predefined Report Engine parameter 3–3, 3–20, 5–5

RB-WINDOW-TITLE Report Engine table field name 4–5

Records
  report engine
    See Report Engine table

Report Code Page In (-rbcpreportin) startup parameter 4–9

Report Engine 1–1
  invoking 4–7

Report Engine parameters 3–2
  guidelines and restrictions 3–24
  incorporating in the report definitions 3–28
  predefined 3–2, 3–26
  prompting for user input 3–26
  RB-BEGIN-PAGE 3–3, 3–4, 5–5
  RB-DB-CONNECTION 3–3, 3–5, 5–5
  RB-DISPLAY ERRORS 3–3, 5–5
  RB-DISPLAY-STATUS 3–4, 3–11, 5–5

RB-END-PAGE 3–3, 3–11, 5–5
RB-FILTER 3–3, 3–11, 5–5
RB-INCLUDE-RECORDS 3–3, 3–13, 5–5
RB-MEMO-FILE 3–3, 3–14, 5–5
RB-NO-WAIT 3–4, 3–14, 5–5
RB-NUMBER-COPIES 3–3, 3–14, 5–5
RB-OUTPUT-FILE 3–3, 3–15, 3–19, 5–5
RB-PRINT-DESTINATION 3–3, 3–15, 5–5
RB-PRINTER-NAME 3–3, 3–16, 5–5
RB-PRINTER-PORT 3–3, 3–17, 5–5
RB-REPORT-LIBRARY 3–3, 3–17, 5–5
RB-REPORT-NAME 3–3, 3–18, 5–5
RB-STATUS 3–18
RB-TAG 3–19
RB-TEST-PATTERN 3–3, 3–20, 5–5
RB-WINDOW-TITLE 3–3, 3–20, 5–5
table override 3–20
user-defined 3–20, 3–26

Report Engine Record Delete (-rbdel) startup parameter 4–10

Report Engine table 4–2
fields 4–2
RB-STATUS field 2–13
records 4–2

Report Engine Table Name (-rbtable) startup parameter 4–13

Report libraries
listing reports 2–2

Report Status file 2–10
RB-STATUS 2–13
RO-ERROR-CODE 2–12
RO-ERROR-MESSAGE 2–12

Report Status File (-rbstatfile) startup parameter 2–10, 4–11

Report Tag (-rbtag) startup parameter 4–13

Report title 3–20

Report Update Status (-rbupds) startup parameter 2–13, 4–14

Reports
generating a list 2–2
pages 2–13
running from an application 5–6
in a batch 5–10
prompting for user input 5–13

reptlist.p procedure 2–3

RO-ERROR-CODE file entry 2–12

RO-ERROR-MESSAGE file entry 2–12

RO-PAGES file entry 2–13

Running multiple reports 4–19

Running reports from an application 4–19

RUNTIME-PARAMETER

## S

Security
_rbpwenc.p procedure 2–9

Startup parameters 4–8

startup.pf file 4–8

Statements
OS-COMMAND 4–16

Status
Report Status file
See Report status file

Syntax notation xi

## T

Table override parameters 3–20

Tables
Report Engine
See Report Engine tables

Test patterns
printing 3–20

Typographical conventions x

## U

UDF Code Page In (-rbcpudfin)
startup parameter 4–10

User-defined parameters 3–26

User-defined Report Engine
parameters 3–20, 3–23