

# Progress Client 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: 4601
Item Number: 81091;9.1C

# Contents

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

**1. Progress Client Deployment Overview** . . . . . **1–1**
1.1 Client Deployment Process . . . . . 1–2
1.2 Client Deployment and Administration Tasks . . . . . 1–2

**2. Managing Client Access to Databases . . . . . 2–1**
2.1 Connecting Clients to Databases . . . . . 2–2
2.1.1 Logical Database Names . . . . . 2–2
2.1.2 Connection Modes . . . . . 2–3
2.1.3 Connection Parameters . . . . . 2–4
2.1.4 Connection Techniques . . . . . 2–5
2.1.5 Connecting to a Non-Progress Database . . . . . 2–12
2.1.6 Connection Denials When the User Count Is Exceeded. . . . . 2–13
2.1.7 Reducing Connection Times . . . . . 2–14
2.1.8 Starting Clients Without Connecting to a Database . . . . . 2–15
2.2 Selecting a Working Database for Client Use . . . . . 2–15
2.2.1 Selecting a Database with the Data Dictionary . . . . . 2–15
2.2.2 Selecting a Database with the Data Administration Tool . . . . . 2–16
2.3 Disconnecting Databases for Clients . . . . . 2–16
2.3.1 Disconnecting a Database with the Data Dictionary . . . . . 2–16
2.3.2 Disconnecting a Database with the Data Administration Tool . 2–17

**3. Maintaining Application Security . . . . . 3–1**
3.1 Compile-time Security . . . . . 3–2
3.1.1 Using Table- and Field-level Security . . . . . 3–2
3.1.2 Setting Table and Field Permissions. . . . . 3–2
3.1.3 Determining the Privileges of the Blank User ID . . . . . 3–6
3.2 Run-time Security . . . . . 3–7
3.3 Operating System Security . . . . . 3–7
3.4 Designating a Security Administrator . . . . . 3–8

**4. Managing Client Performance . . . . . 4–1**
4.1 Procedure Loading and Execution . . . . . 4–2
4.1.1 Using R-code Libraries to Improve R-code Performance . . . . . 4–2
4.1.2 R-code Execution Environment. . . . . 4–3
4.1.3 Monitoring R-code Activity . . . . . 4–6
4.2 Distributing Progress Files on a Network . . . . . 4–9
4.2.1 Distributing Progress System Files . . . . . 4–9
4.2.2 Distributing Procedure Files . . . . . 4–9
4.2.3 Distributing Temporary Files . . . . . 4–10
4.2.4 Distributing Log Files. . . . . 4–10
4.3 Temporary File I/O . . . . . 4–11
4.4 Sorting I/O . . . . . 4–12

**5. Maintaining User Environments . . . . . 5–1**
5.1 Maintaining the Windows User Environment . . . . . 5–2
5.1.1 Using the INI2REG Utility . . . . . 5–2
5.1.2 Searching progress.ini and the Registry at Startup. . . . . 5–5
5.1.3 Maintaining the progress.ini File . . . . . 5–8

5.1.4 Windows Icons for Running Progress . . . . . 5–33
5.1.5 Starting the Progress Client . . . . . 5–34
5.1.6 Modifying Progress Client Icons . . . . . 5–34
5.2 Maintaining the UNIX User Environment . . . . . 5–34
5.2.1 Maintaining the buildenv Script . . . . . 5–35
5.2.2 Maintaining the PROTERMCAP File . . . . . 5–36
5.2.3 PROTERMCAP Syntax . . . . . 5–37
5.2.4 Terminal Name Entries . . . . . 5–40
5.2.5 Terminal Capabilities Entries . . . . . 5–41
5.2.6 Vermont Views Key Function Capabilities . . . . . 5–47
5.2.7 Pointer to Key Functions . . . . . 5–49
5.2.8 Specifying Colors . . . . . 5–50
5.2.9 Specifying the Cursor Motion Capability . . . . . 5–51
5.2.10 Specifying Keyboard Mappings . . . . . 5–53
5.2.11 Extended Character Support Entries . . . . . 5–61
5.2.12 Creating a PROTERMCAP Entry . . . . . 5–62
5.2.13 Testing Terminal Entries . . . . . 5–62
5.2.14 Setting Up the Terminal . . . . . 5–63
5.2.15 Setting the Terminal Type . . . . . 5–63
5.2.16 Setting the PROTERMCAP Environment Variable . . . . . 5–64
5.2.17 Reverting to the Default PROTERMCAP File . . . . . 5–64
5.2.18 Copying an Existing Terminal Entry . . . . . 5–64

**6. Managing R-code Libraries . . . . . 6–1**
6.1 Library Overview . . . . . 6–2
6.1.1 Loading R-code from a File . . . . . 6–3
6.1.2 Loading R-code from a Standard Library . . . . . 6–3
6.1.3 Loading R-code from a Memory-mapped Library . . . . . 6–4
6.2 Setting Up a Library . . . . . 6–5
6.3 Running Procedures from a Library . . . . . 6–6
6.3.1 Using an Absolute Pathname . . . . . 6–6
6.3.2 Using a Relative Pathname . . . . . 6–6
6.3.3 Related Functions . . . . . 6–7
6.4 Libraries and PROPATH . . . . . 6–8
6.5 Memory and Network Considerations . . . . . 6–9
6.6 Using the PROLIB Utility . . . . . 6–9
6.6.1 Using Wild Cards . . . . . 6–11
6.6.2 Creating a Standard Library . . . . . 6–12
6.6.3 Generating a Memory-mapped Library . . . . . 6–13
6.6.4 Adding Files to a Standard Library . . . . . 6–14
6.6.5 Replacing Files in a Standard Library . . . . . 6–15
6.6.6 Deleting Files from a Standard Library . . . . . 6–15
6.6.7 Listing the Contents of a Library . . . . . 6–16
6.6.8 Extracting Files from a Standard Library . . . . . 6–17

6.6.9 Extracting Files to Your Current Directory . . . . . 6–18
6.6.10 Compressing a Standard Library . . . . . 6–19
6.6.11 PROLIB Command Examples . . . . . 6–20
6.7 Code Page Compatibility . . . . . 6–21

**7. Managing Print Devices . . . . . 7–1**
7.1 Printing in Progress . . . . . 7–2
7.1.1 OUTPUT TO Statement . . . . . 7–3
7.1.2 OUTPUT THROUGH Statement . . . . . 7–4
7.2 Printing on UNIX . . . . . 7–5
7.2.1 UNIX Networks and Printing . . . . . 7–6
7.3 Printing in Windows . . . . . 7–8
7.4 Setting Up Output-routing . . . . . 7–8
7.4.1 Using a Startup Procedure . . . . . 7–9
7.4.2 Using an Output-routing Table . . . . . 7–10
7.4.3 Using Interactive Output-routing . . . . . 7–12
7.5 Configuring a Printer . . . . . 7–12
7.5.1 Creating a Printer-control Table . . . . . 7–13

**A. Building Progress Executables . . . . . A–1**
A.1 The Build Process . . . . . A–2
A.2 Software and Hardware Requirements . . . . . A–4
A.3 Licensing Requirements . . . . . A–4
A.4 PROBUILD Overview . . . . . A–4
A.5 Building Executables on UNIX . . . . . A–5
A.5.1 Preparing the UNIX Software Environment . . . . . A–5
A.5.2 Running PROBUILD on UNIX . . . . . A–6
A.5.3 Linking on UNIX . . . . . A–13
A.6 Building Executables in Windows . . . . . A–13
A.6.1 Preparing the Windows Software Environment . . . . . A–14
A.6.2 Running PROBUILD in Windows . . . . . A–15
A.6.3 Linking in Windows . . . . . A–24
A.7 Troubleshooting . . . . . A–24

**B. Progress Application Limits . . . . . B–1**
B.1 Input/Output Limits . . . . . B–2
B.2 Sorting Limits . . . . . B–2
B.3 Name Limits . . . . . B–3
B.4 Compiler Limits . . . . . B–4

**Index . . . . . Index–1**

**Figures**


| | | |
|---|---|---|
| Figure 2–1: | Connection Denial When the User Count Is Exceeded | 2–13 |
| Figure 4–1: | Sample Statistics (-y) Output | 4–7 |
| Figure 4–2: | Sample Segment Statistics (-yd) Output | 4–8 |
| Figure 5–1: | Windows Environment Information Search Path | 5–6 |
| Figure 5–2: | Sample Progress Initialization File in Windows | 5–8 |
| Figure 5–3: | Sample buildenv Script on UNIX | 5–35 |
| Figure 5–4: | Parts of a PROTERMCAP Entry | 5–37 |
| Figure 6–1: | Loading R-code from a File | 6–3 |
| Figure 6–2: | Loading R-code from a Standard Library | 6–4 |
| Figure 6–3: | Loading R-code from a Memory-mapped Library | 6–5 |
| Figure 7–1: | Sample Application Startup Procedure (UNIX) | 7–9 |
| Figure 7–2: | Sample Application Startup Procedure (NT) | 7–10 |
| Figure 7–3: | Sample Output-routing Table | 7–11 |
| Figure 7–4: | Sample Printer-control Table | 7–14 |
| Figure A–1: | Build Process Steps | A–2 |

**Tables**


Table 3–1: Security Permissions . . . . . 3–4
Table 3–2: Access Restrictions for Tables . . . . . 3–4
Table 4–1: Startup Parameters for Tuning the Execution Environment . . . . . 4–4
Table 4–2: Startup Parameters for Tuning R-code Libraries . . . . . 4–5
Table 4–3: Startup Parameters for Monitoring R-code Activity . . . . . 4–6
Table 4–4: Temporary Client Session Files . . . . . 4–11
Table 4–5: Startup Parameters for Tuning Sort Performance . . . . . 4–12
Table 5–1: DLL Files for AS/400 Communications . . . . . 5–10
Table 5–2: User-definable Key Bindings in Windows . . . . . 5–26
Table 5–3: Static Key Bindings in Windows . . . . . 5–28
Table 5–4: User-definable Key Bindings in Windows . . . . . 5–29
Table 5–5: Environment Variables . . . . . 5–32
Table 5–6: Windows Installed Icons . . . . . 5–33
Table 5–7: Syntax for Specifying String Values . . . . . 5–39
Table 5–8: Alphabetical Listing of Capability Mnemonics . . . . . 5–43
Table 5–9: Data Types and Operators . . . . . 5–46
Table 5–10: Vermont Views Key Function Mnemonics . . . . . 5–48
Table 5–11: Syntax for Specifying Cursor Motion String Value . . . . . 5–52
Table 5–12: Progress PROTERMCAP Key Functions . . . . . 5–53
Table 5–13: Setting the Terminal Type . . . . . 5–63
Table 5–14: Setting the PROTERMCAP Environment Variable . . . . . 5–64
Table 5–15: Reverting to the Default PROTERMCAP File . . . . . 5–64
Table 6–1: PROLIB Utility Parameters . . . . . 6–10
Table 6–2: List (-list) Parameter Library Information . . . . . 6–16
Table 6–3: List (-list) Parameter File Information . . . . . 6–16
Table 7–1: Standard Printer Control Sequences . . . . . 7–12
Table A–1: PROBUILD Software Requirements . . . . . A–4
Table A–2: PROBUILD Product List on UNIX . . . . . A–7
Table A–3: Configurable Elements on UNIX . . . . . A–11
Table A–4: PROBUILD Product List in Windows . . . . . A–17
Table A–5: Configurable Elements in Windows . . . . . A–22
Table B–1: Input/Output Limits . . . . . B–2
Table B–2: Sorting Limits . . . . . B–2
Table B–3: Name Limits . . . . . B–3
Table B–4: Compiler Limits . . . . . B–4

# Preface

## Purpose

This guide describes the Progress 4GL client application deployment process and application administration concepts and procedures.

## Audience

This guide is intended for Progress application administrators.

## Organization of This Manual

Chapter 1, “Progress Client Deployment Overview”

Presents an overview of the client deployment process, and outlines the deployment and administration tasks you must perform.

Chapter 2, “Managing Client Access to Databases”

Describes how to connect and disconnect databases for Progress clients.

Chapter 3, “Maintaining Application Security”

Describes how Progress implements application security, including designating security administrators and setting table and field permissions.

Chapter 4, “Managing Client Performance”

Describes how to monitor and tune Progress client application performance.

Chapter 5, “Maintaining User Environments”

Describes how to maintain Progress user interface environments in Windows and on UNIX.

Chapter 6, “Managing R-code Libraries”

Describes Progress r-code libraries and how to use the PROLIB utility.

Chapter 7, “Managing Print Devices”

Describes how to set up print devices on Progress-supported operating systems.

Appendix A, “Building Progress Executables”

Describes the process and requirements for building customized Progress executables. It also provides instructions for using PROBUILD in Windows and on UNIX.

Appendix B, “Progress Application Limits”

Describes the Progress limits you must consider when developing a Progress client application.

## How to Use This Manual

This guide assumes that you are familiar with the Progress 4GL client application development and deployment environments, as well as Progress-supported operating systems. This guide contains step-by-step instructions for accomplishing specific tasks and has cross-references to the books, chapters, or sections where you can find detailed or related information about a particular topic.

## 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. On the Windows platform, 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**).

On the UNIX platform, you can use the Progress PRO command to start a single-user mode character Progress client session and view a brief description of a message by providing its number. Follow these steps:

1 ♦ Start the Progress Procedure Editor:

```
install-dir/dlc/bin/pro
```

2 ♦ Press **F3** to access the menu bar, then choose Help→ Messages.

3 ♦ Type the message number, and press **ENTER**. Details about that message number appear.

4 ♦ Press **F4** to close the message, press **F3** to access the Procedure Editor menu, and choose File→ Exit.

## 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 Deployment Guide* (Windows only)

An administration and development guide for generating Report Builder reports using the Progress Report Engine.

*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 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 Version 3 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

# Progress Client Deployment Overview

Deploying a Progress 4GL client application (hereinafter referred to as “client” or “application”) is a process that involves developing an application and installing that application in end-user environments. Once you have deployed your application, you must manage the end-user environments and system resources for your users.

This chapter presents an overview of the client deployment process, and outlines the deployment and administration tasks you must perform. This chapter provides information about the following topics:


- “Client Deployment Process”
- “Client Deployment and Administration Tasks”

## 1.1 Client Deployment Process

While deployment requirements vary between applications, you follow a basic deployment process to deploy any Progress 4GL client application.

The client deployment process consists of three basic phases:

1. **Planning** — The phase in which you identify your deployment requirements and define your deployment strategy.
2. **Development** — The phase in which you design, develop, and debug your client application.
3. **Installation and Configuration** — The phase in which you deliver your client application to end-users, and set up the end-user environment.

Each phase of the client deployment process has its own set of requirements and dependencies that you must consider when defining your client deployment strategy. Decisions made in an earlier phase will affect the following phases and the process as a whole.

Progress Software Corporation provides you with the following services to help you identify your client deployment options and accomplish the client deployment process for your application:

- The Progress Software Consulting Services organization which can assist you in all phases of the client deployment process.
- The Progress Technical Services staff which can answer your questions about the ADE tools, 4GL usage, and database administration.

## 1.2 Client Deployment and Administration Tasks

Following are some of the Progress 4GL client application deployment and administration tasks you must perform:

- Manage client access to databases
- Maintain application security
- Manage client performance
- Maintain user environments

- Manage r-code libraries
- Manage print devices

The remaining chapters in this guide provide detailed information about each of these tasks, as well as the procedures and utilities you use to accomplish them. The appendixes provide useful information about building Progress executables and Progress application limits.

**NOTE:** Certain deployment and administration tasks will differ among operating systems.

For information about designing and coding a Progress 4GL client application, see the *Progress Programming Handbook*.

For information about creating and maintaining a Progress database, see the *Progress Database Administration Guide and Reference*.

For information about AppServer deployment and administration tasks, see the *Building Distributed Applications Using the Progress AppServer* manual.

# 2

# Managing Client Access to Databases

This chapter describes how to connect and disconnect databases for Progress clients. It provides information about the following topics:


- "Connecting Clients to Databases"
- "Selecting a Working Database for Client Use"
- "Disconnecting Databases for Clients"

## 2.1 Connecting Clients to Databases

To access a database, you must connect to the database. You can connect to one or more databases during a single Progress session. You can also start a Progress session without connecting to a database, and connect later.

**NOTE:** On UNIX, the kernel configuration's file descriptor limit imposes a practical limit on connected databases. "Unable to open file" or "Unable to allocate I-Node" messages often mean your system's file descriptors have been exhausted. "Unable to allocate shared memory" often means your shared memory has been exhausted on your system. For more information about kernel configurations, see the *Progress Installation and Configuration Guide Version 9 for UNIX*.

When connecting to databases, consider the following factors:

- Logical database names
- Connection modes
- Connection parameters
- Connection techniques
- Connecting to a non-Progress database
- Connection denials when the user count is exceeded
- Reducing connection times
- Starting clients without connecting to a database

### 2.1.1 Logical Database Names

A *logical database name* is a database reference that represents the name of a connected physical database. Progress uses the logical database name to resolve database references. When a procedure is compiled against a database, Progress stores the logical database name in the procedure's r-code. When a procedure executes, its database references must match the logical name of a connected database.

A database has a logical name whenever it is connected. When you connect to a database, Progress automatically assigns a default logical name for the current session. The default logical name consists of the physical database name without the .db suffix. For example, if you connect to a database with the physical name `mydb1.db`, the default logical database name is `mydb1`. Alternatively, you can use the Logical Database Name (-ld) startup parameter to specify a logical database name other than the default.

For example, to assign the logical name firstdb to the physical database `mydb1.db` using the PRO command, enter the command using the following syntax:

```
pro mydb1 -ld firstdb
```

Logical database names allow you to run procedures on different physical databases without recompiling. To run a compiled procedure on a second physical database, the database must meet the following criteria:

- It must have the identical structure (data definitions) and table time stamps as the database you compiled the procedure against. However, if you use CRC-based object code (the default), the database's table time stamps do not affect the ability of the code to run on the database. The structure of the database must be the same. For more information about CRC-based object code, see the *Progress Programming Handbook*.
- It must be connected with the same logical name as the database you compiled the procedure against.

A database connection fails if the logical database name of the database that you are connecting has the same logical name as an already connected database. If you try to connect a database with a logical database name that matches the logical database name of an existing connected database of the same database type (Progress, ORACLE, etc.), Progress assumes that database is connected and ignores the request.

You can change a database's logical name in a Progress procedure with the CREATE ALIAS statement. For more information about logical and physical database names and aliases, and the CREATE ALIAS statement, see the *Progress Language Reference* and the *Progress Programming Handbook*.

### 2.1.2 Connection Modes

A database is connected in one of three connection modes:

- **Single-user** — Only the current Progress session can access the specified database. If the database is already in use by another user, you cannot connect to the database from the current Progress session.
- **Multi-user client** — The current Progress session accesses the database through a server process. Multi-user client is the default connection mode on non-shared-memory systems and when you use the Host Name (-H) and Service Name (-S) parameters on shared-memory systems. All remote (over a network) connections access the database through a database server and are in multi-user client mode.

- **Multi-user direct access** — The current Progress session accesses the database directly using shared memory. A process connected in multi-user direct-access mode is a self-service client. Direct access is available only on shared-memory systems and is the default connection mode on these systems. For information about shared-memory configurations, see the *Progress Installation and Configuration Guide Version 9 for Windows* or the *Progress Installation and Configuration Guide Version 9 for UNIX*.

  You cannot connect to remote databases in multi-user direct-access mode.

When you connect to a database with a Progress multi-user startup command, each database you specify on the command line is connected in the default multi-user mode. Each database you specify in a startup file or in a CONNECT statement in an application is also connected in the default multi-user mode for the current machine.

The first database specified after the Progress single-user or Progress single-user batch command is connected in single-user mode by default. Any additional databases specified with the Database Name (-db) connection parameter are connected in the default multi-user mode for the system. Use the Single-user Mode (-1) startup parameter to override the default multi-user connection mode for a database and connect to the database in single-user mode.

For more information about the Host Name (-H), Service Name (-S), Database Name (-db), and Single-user Mode (-1) startup parameters, see the *Progress Startup Command and Parameter Reference*.

### 2.1.3 Connection Parameters

*Connection parameters* control how you connect to databases. They are a subset of the Progress startup parameters. All of the database connection techniques provide a way for you to specify connection parameters.

You can use a parameter file to specify connection parameters with any of the connection techniques. A *parameter file* is an operating system file that contains one or more startup parameters. A parameter file has a .pf extension. Use the Parameter File (-pf) startup parameter to invoke a parameter file.

Parameter files are an important factor in developing database connection strategies for multi-database applications. For example, you can use parameter files to maintain the startup parameters for a particular database, group of users, or system configuration.

For more information about startup parameters and parameter files, see the *Progress Startup Command and Parameter Reference*.

### 2.1.4 Connection Techniques

There are five ways to connect a database:

- As an argument when starting Progress
- With the CONNECT statement (in the Progress Procedure Editor or in a Progress procedure)
- With the Progress Data Dictionary
- With the Progress Data Administration tool
- Using the Auto-connect feature

**NOTE:** Starting a server or broker for a database is distinct from connecting to a database. You must start a server or broker for a database before you can connect to the database in multi-user mode. For information about starting a server or broker, see the *Progress Installation and Configuration Guide Version 9 for Windows* or the *Progress Installation and Configuration Guide Version 9 for UNIX*.

This section explains how to connect to databases using the various techniques. The *Progress Programming Handbook* explains why you might choose one technique instead of another.

#### Connecting at Startup

You can connect to databases at session startup with one of the Progress startup commands or with an application startup file. For example, the following PRO command starts a single-user session and connects to three databases:

```
pro -db mydb1 -db mydb2 -db mydb3
```

**NOTE:** To retain compatibility with earlier versions, Progress does not require you to specify the Database Name (-db) connection parameter for the first database on the command line.

If you specify more than one database when you start a session, it is important to specify the connection parameters for each database directly after the database name to which they apply, and before the next Database Name (-db) connection parameter. Progress applies database connection parameters only to the previously specified database.

The following syntax example illustrates the correct location for database connection parameters:

```
mpro [ -db ] db1-name [ db1-parameters ]
    [ -db db2-name [ db2-parameters ] ] ...
```

You can specify all other parameters anywhere on the command line. If you specify the same parameter more than once, Progress uses the value you specified for the last instance of the parameter.

You can also use a parameter file to specify database connection parameters. The following example shows a UNIX startup script for a Progress application. The last line invokes the `_progres` executable:

```
DLC=${DLC-/usr/dlc}; export DLC
PATH=:$PATH:$DLC; export PATH
PROPATH=:$DLC; export PROPATH
exec $DLC/_progres -pf parm1.pf -pf parm2.pf -p $APPL/start.p
```

The previous startup script sets up environment variables for a Progress application and starts a Progress application session using two parameter files called `parm1.pf` and `parm2.pf`. The `parm1.pf` parameter file contains the following entry:

```
-db appldb1 -1 bi-file1
```

The `parm2.pf` parameter file contains the following entry:

```
-db appldb2 -1 bi-file2
```

This example illustrates a common approach to connecting multiple databases when you start Progress using a parameter file for each database. Each parameter file is specified on the command line with the Parameter File (-pf) startup parameter.

### Connecting from a Procedure with the CONNECT Statement

The CONNECT statement allows you to connect to a database from a Progress procedure or from the Progress Procedure Editor.

The following CONNECT statements connect the appldb1 database in single-user mode, the appldb2 database in multi-user mode, and the databases specified in the parameter file, `parm3.pf` (respectively):

```
CONNECT appldb1 -1.
CONNECT appldb2.
CONNECT -pf parm3.pf.
```

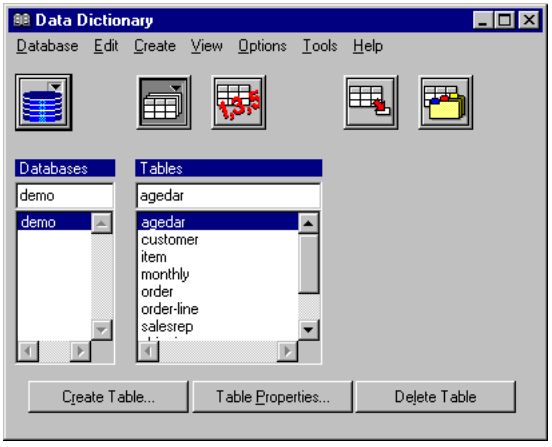
Connecting with the CONNECT statement is very similar to connecting at startup. Although it is possible to connect to several databases within one CONNECT statement, it is a good idea to connect only one database per CONNECT statement. A connection failure for one database causes a termination of the current CONNECT statement, leaving any subsequent database specified with the CONNECT statement unconnected. For detailed information about the CONNECT statement, see the *Progress Language Reference* and the *Progress Programming Handbook*.

### Connecting with the Data Dictionary

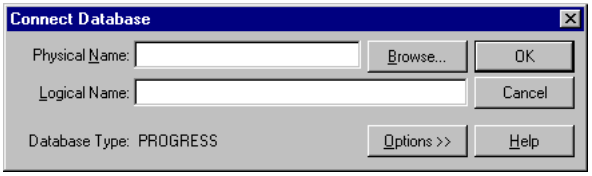
You can connect to a database during a Progress session using the Data Dictionary.

Follow these steps to connect to a database with the Data Dictionary using a graphical user interface:

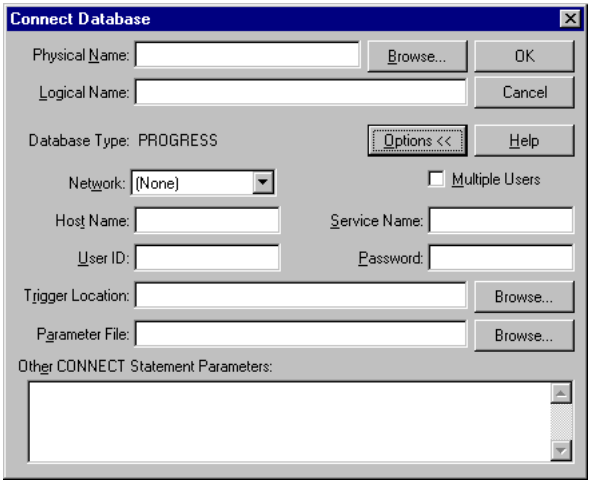
1 ♦ Access the Data Dictionary. The Data Dictionary main window appears:



2 ♦ Choose Database→ Connect. The Connect Database dialog box appears:




3 ♦ Choose the Options button. The Connect Database dialog box expands to show optional connection controls:




4 ♦ Enter the following information, then choose OK:

- **Physical Name** — Specifies the actual name of the database on a disk.
- **Logical Name** — Specifies the database name that references a connected physical database.
- **Database Type** — Specifies the database type. The only possible value is Progress.
- **Network** — Specifies the network type, if you are connecting to a network server. The only possible values are TCP or SNA (for Progress/400).
- **Multiple Users** — Specifies whether you want multiple users to be able to access this database simultaneously.
- **Host Name** — Specifies the database server machine in a network environment.
- **Service Name** — Specifies the broker or server service name in a network environment.

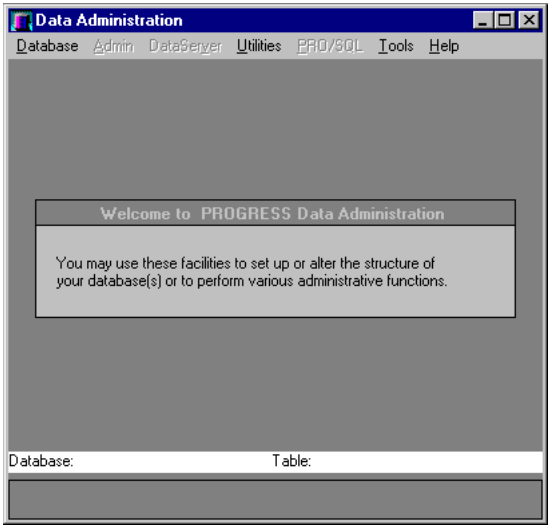
- **User Id** — Identifies your user ID.
- **Password** — Identifies your password.
- **Trigger Location** — Identifies a directory or Progress library where trigger code is stored.
- **Parameter File** — Specifies the parameter filename that contains the startup parameters for the database.
- **Other CONNECT Statement Parameters** — Specifies any other startup parameters for the database that are not included in the parameter file.

Progress returns you to the Data Dictionary main window. The Data Dictionary constructs (and executes) a CONNECT statement using the information you supply. Therefore, any rules that apply to the CONNECT statement also apply to database connections using the Data Dictionary.

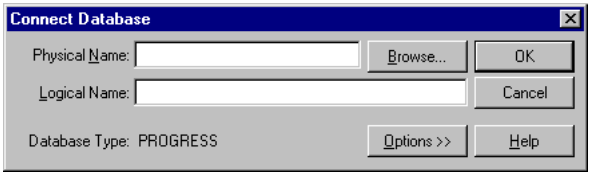
### Connecting with the Data Administration Tool

Follow these steps to connect to a database with the Data Administration tool using a graphical interface:

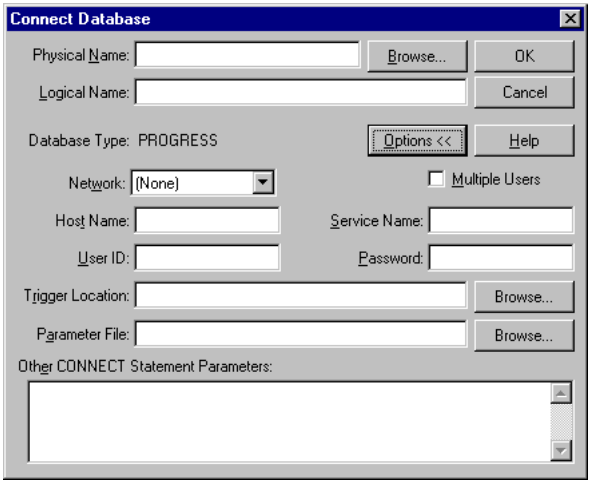
1 ♦ Access the Data Administration tool. The Data Administration main window appears:



2 ♦ Choose Database→ Connect. The Connect Database dialog box appears:




3 ♦ Choose the Options button. The Connect Database dialog box expands to show optional connection controls:




4 ♦ Enter the following information, then choose OK:

- **Physical Name** — Specifies the actual name of the database on a disk.
- **Logical Name** — Specifies the database name that references a connected physical database.
- **Database Type** — Specifies the database type. The only possible value is Progress.
- **Network** — Specifies the network type, if you are connecting to a network server. The only possible values are TCP or SNA (for Progress/400).
- **Multiple Users** — Specifies whether you want multiple users to be able to access this database simultaneously.
- **Host Name** — Specifies the database server machine in a network environment.
- **Service Name** — Specifies the broker or server service name in a network environment.

- **User Id** — Identifies your user ID.
- **Password** — Identifies your password.
- **Trigger Location** — Identifies a directory or Progress library where trigger code is stored.
- **Parameter File** — Specifies the parameter filename that contains the startup parameters for the database.
- **Other CONNECT Statement Parameters** — Specifies any other startup parameters for the database that are not included in the parameter file.

Progress returns you to the Data Administration main window. The Data Administration tool constructs (and executes) a CONNECT statement using the information you supply. Therefore, any rules that apply to the CONNECT statement also apply to database connections using the Data Administration tool.

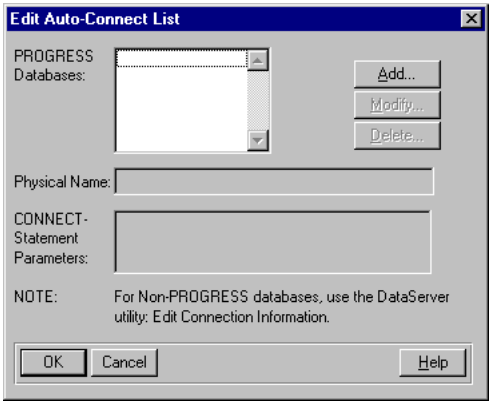
### Connecting with Auto-connect

The Progress auto-connect feature allows you to connect to databases automatically as required during program execution. To perform an auto-connect operation, Progress uses information stored in a primary database to connect to a secondary application database, before running compiled procedures that access the second database. The primary database (the one that contains the auto-connect information for one or more additional databases) must be connected before Progress can perform an auto-connect.

Use the Data Administration tool to build an auto-connect list for a primary database. Follow these steps to create or edit an auto-connect list:

1 ♦ Access the Data Administration tool.

2 ♦ Choose Utilities→ Edit Progress Auto-Connect List. The Edit Auto-Connect List dialog box appears:




3 ♦ Enter or edit connection information for a database.

If you connect to a database with the CONNECT statement, and that database also has an auto-connect entry in an already connected database, the connection information from both the CONNECT statement and the auto-connect list is merged. In this situation, the connection information in the CONNECT statement takes precedence. For more information about the CONNECT statement, see the *Progress Language Reference*.

If the primary database is a schema holder for a DataServer schema, the DataServer schema is automatically included in the auto-connect list, but it does not appear on the auto-connect list as viewed from the Data Dictionary.

### 2.1.5 Connecting to a Non-Progress Database

Follow these steps to connect to a non-Progress database (ORACLE, for example):

1 ♦ Create a Progress database. This database stores the target database's structure as a Progress schema; it is a *schema holder*. A schema that corresponds to a non-Progress database is a *DataServer schema*.

2 ♦ Connect the schema holder.

3 ♦ Use the appropriate Create Schema parameter from the Data Dictionary's database menu to create the DataServer schema in the schema holder. This parameter:

- Creates a _Db table record (auto-connect entry) for the non-Progress database in the schema holder database
- Connects to the non-Progress schema (through a DataServer)
- Displays the menu so that you can choose the tables that you want to include in the Progress schema

4 ♦ Connect the DataServer schema.

For complete information about connecting and using non-Progress databases, see the Progress DataServer guides (the *Progress DataServer for ODBC Guide*, the *Progress DataServer for ORACLE Guide*, or the *Progress/400 Product Guide*).

### 2.1.6 Connection Denials When the User Count Is Exceeded

When a user attempts to connect to a database using either a CONNECT statement or a command and exceeds the maximum number of users allowed to connect to the database, as specified with the Number of Users (-n) startup parameter, Progress denies the connection request and displays an error message. Progress also records the connection denial in the database log file.

Figure 2–1 shows a connection denial when a connection request exceeds the maximum number of users. In this example, the Number of Users (-n) startup parameter allows a maximum of three user connections. When the fourth user attempts to connect, Progress denies the connection.

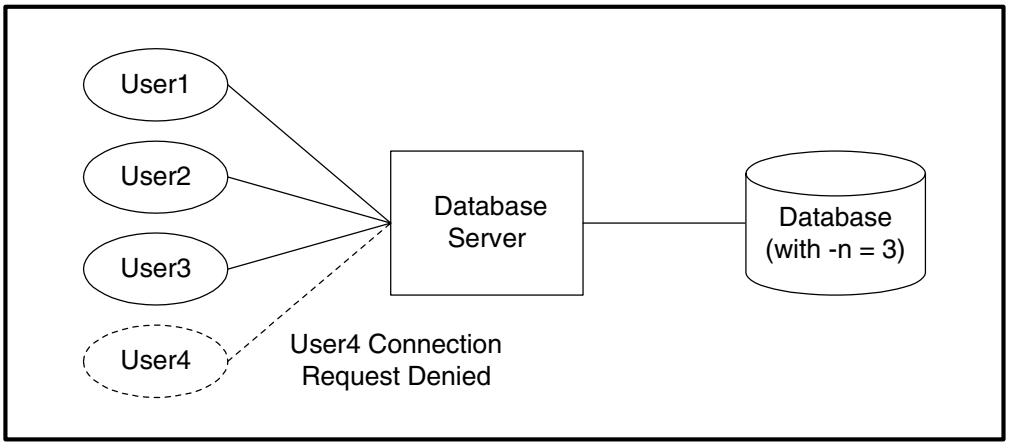



**Figure 2–1: Connection Denial When the User Count Is Exceeded**

When Progress denies a user connection because the maximum number of users is exceeded, you must terminate an existing connection before the user can establish a new connection.

You can use database usage reporting features to track user connections to databases. For information about database usage reporting features, see the *Progress Database Administration Guide and Reference* and either the *Progress Installation and Configuration Guide Version 9 for Windows* or the *Progress Installation and Configuration Guide Version 9 for UNIX*.

For more information about the Number of Users (-n) startup parameter, see the *Progress Startup Command and Parameter Reference*.

For information about coding applications to handle connection denials, see the *Progress Programming Handbook*.

### 2.1.7 Reducing Connection Times

To perform database activities, the Progress client keeps a copy of the database schema, called the *schema cache*, in memory. By default, Progress creates the schema cache by reading the database schema stored in the database file. The client reads the schema in two parts:

- During connection, the client reads all the _Db and _File records.
- When the database is referenced, the client reads all the _Index, _Index-field, _File-trig, and _Field-trig records.

The time required to read the schema is usually minimal. However, the time required to read the schema might be unacceptable under the following conditions:

- If the client connects to the database over a wide area network.
- When a large number of clients connect to a database simultaneously. For example, after a database shutdown or crash.

To reduce the amount of time required to connect, Progress lets you store the schema cache as a binary file, called a *schema cache file*, on a local disk. The client can then read the schema directly from the schema cache file.

To create the schema cache file, you build the desired schema cache and save it to a binary file using the Progress statement SAVE CACHE. The schema cache file is portable across systems, so you can create the file once and distribute it across a heterogeneous network of systems. For information about building and saving the schema cache file, see the *Progress Programming Handbook*. For information about the SAVE CACHE statement, see the *Progress Language Reference*.

To connect using the local schema cache file, specify the Schema Cache File (-cache) startup parameter when you connect to a database. If the schema cache file is valid, Progress reads the schema from the local file instead of from the database. The schema cache is valid if the time stamp of the schema cache file matches the time stamp in the database master block. If the time stamps do not match, or for some reason Progress cannot read the file, Progress issues a warning message and reads the schema from the database. For information about the Schema Cache File (-cache) startup parameter, see the *Progress Startup Command and Parameter Reference*.

### 2.1.8 Starting Clients Without Connecting to a Database

To permit maximum flexibility for your application, you might want to start Progress without connecting to a database. For example, you can start Progress and name a startup procedure, such as a main menu procedure. This startup procedure (or a procedure that it calls) can connect to a database based on an end user's menu selection. To start Progress without connecting to a database, execute the Progress startup command without specifying a database name. On UNIX, use the PRO command; in Windows, use the PROWIN32 command. For more information about startup commands, see the *Progress Startup Command and Parameter Reference*.

## 2.2 Selecting a Working Database for Client Use

A working database is the database that you are currently accessing through the Data Dictionary or the Database Administration tool. The working database is also known as the active database. You can select a working database with the Data Dictionary when using either a graphical or character interface, and with the Data Administration tool only when using a graphical interface.

### 2.2.1 Selecting a Database with the Data Dictionary

Follow these steps to select a working database with the Data Dictionary when using a graphical interface:

1 ♦ Access the Data Dictionary. The Data Dictionary displays the connected databases and their tables and fields.

2 ♦ Select the database you want for the working database.

Follow these steps to select a working database with the Data Dictionary when using a character interface:

1 ♦ Access the Data Dictionary. The Data Dictionary appears.

2 ♦ Choose Database→ Select a Working Database. Progress lists the connected databases.

3 ♦ Select the new working database, then choose Enter to return to the Data Dictionary main window.

### 2.2.2 Selecting a Database with the Data Administration Tool

Follow these steps to select a working database with the Data Administration tool when using a graphical interface:

1 ♦ Access the Data Administration tool.

2 ♦ Choose Database→ Select a Working Database. Progress lists the connected databases.

3 ♦ Select the new working database, then choose OK to return to the Data Administration main window.

## 2.3 Disconnecting Databases for Clients

By default, Progress disconnects all connected databases at the end of a session. To explicitly disconnect a database from a Progress application, use the DISCONNECT statement. For more information about the DISCONNECT statement, see the *Progress Language Reference* and the *Progress Programming Handbook*.

**NOTE:** When you have more than one database connected and you disconnect the working database, Progress automatically switches to the next connected database on the list.

You can disconnect a database with the Data Dictionary when using either a graphical or character interface, and with the Data Administration tool only when using a graphical interface.

### 2.3.1 Disconnecting a Database with the Data Dictionary

Follow these steps to disconnect a database with the Data Dictionary when using a graphical interface:

1 ♦ Access the Data Dictionary. The Data Dictionary displays the connected databases and their tables and fields.

2 ♦ Select the database you want to disconnect from the database selection list.

3 ♦ Choose Database→ Disconnect.

4 ♦ At the prompt, verify that you want to disconnect the database.

Follow these steps to disconnect a database with the Data Dictionary when using a character interface:

1 ♦ Access the Data Dictionary.

2 ♦ Choose Database→ Disconnect. Progress lists the connected databases.

3 ♦ Select the database you want to disconnect, then choose Enter.

4 ♦ At the prompt, verify that you want to disconnect the database.

### 2.3.2 Disconnecting a Database with the Data Administration Tool

Follow these steps to disconnect a database with the Data Administration tool when using a graphical interface:

1 ♦ Access the Data Administration tool.

2 ♦ Choose Database→ Disconnect. Progress lists the connected databases.

3 ♦ Select the database you want to disconnect, then choose OK.

4 ♦ At the prompt, verify that you want to disconnect the database.

# 3

# Maintaining Application Security

Progress provides two types of security: *application security* which prevents unauthorized users from running application procedures or accessing data in a database, and *database file security* which prevents unauthorized users from modifying or removing database files.

Progress provides four levels of security: *compile-time security* ensures that only authorized users can compile procedures that perform specific data accesses, *run-time security* ensures that only authorized users can run specific precompiled procedures, *connection security* ensures that only authorized users can connect to the database, and *schema security* ensures that only authorized users can modify table, field, and index definitions.

Progress also relies on security mechanisms at the operating system level to ensure that only authorized users access r-code procedure files, procedure library files, and database files.

This chapter provides information about the following topics:


- "Compile-time Security"
- "Run-time Security"
- "Operating System Security"
- "Designating a Security Administrator"


For information about establishing and maintaining connection security, schema security, and database file security, see the *Progress Database Administration Guide and Reference*.

## 3.1 Compile-time Security

Compile-time security checking is built into Progress. You define compile-time security for an application database at the table and field levels to prevent the user from writing their own procedures to access data in the database.

Progress lets you define the type of access rights or permissions different users can have to the tables and fields in your database applications. Progress checks these permissions when the user runs and compiles a procedure for the first time during a Progress session. However, permissions are not checked when the user runs procedures that are precompiled. For information about how to enable security for precompiled procedures, see the "Run-time Security" section later in this chapter.

If you use CRC-based r-code (the default), the user can compile a procedure against a database that has the same schema as the database (a counterfeit database) and then run the procedure against the database. Since Progress does not check the permissions of the database at compile time, you have no compile-time security. However, you can use the PROUTIL utility's DBAUTHKEY qualifier to set an authorization key for the database. The authorization key prevents unwanted r-code from being run against the database.

For more information about CRC-based object code, see the *Progress Programming Handbook*. For more information about the PROUTIL utility, see the *Progress Database Administration Guide and Reference*.

### 3.1.1 Using Table- and Field-level Security

After you have designated yourself and others as security administrators, be sure that the user ID of each security administrator is included in all the permissions lists for the tables and fields for the database. For example, if you assign user ID sjones as a security administrator, but sjones is not included in the permissions for the customer table, sjones will not be able to modify permissions for the customer table. Any attempt to do so results in an error message "Invalid access to change security for customer table." Thus, to modify table and field permissions, a user ID must be designated as a security administrator **and** be included in each of the individual table and field permissions.

For information about designating a security administrator, see the "Designating a Security Administrator" section later in this chapter.

### 3.1.2 Setting Table and Field Permissions

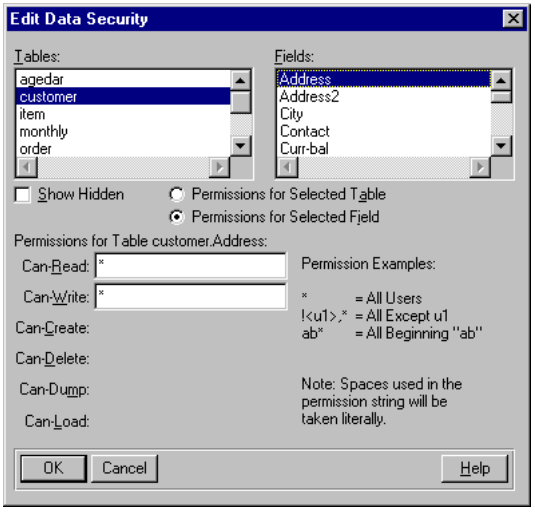
You set table and field permissions by using the Change/Display Data Security utility in the Data Dictionary. Note that once security administrators are designated, only they can access this utility. All other users are denied access with the message "You must be a Security Administrator to execute this function."

Follow these steps to set table or field permissions:

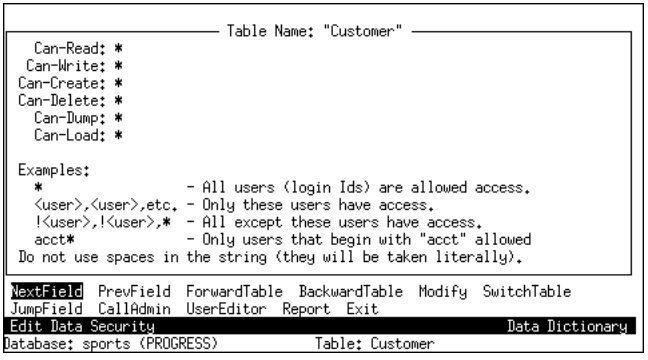
1 ♦ Access the Data Administration tool if you are using a graphical interface or the Data Dictionary if you are using a character interface.

2 ♦ Choose Admin→ Security→ Edit Data Security.

If you are using a graphical interface, the Edit Data Security dialog box appears:




If you are using a character interface, Progress alphabetically lists the tables defined for the working database. Choose the table for which you want to specify permissions. The Edit Data Security dialog box appears:



3 ♦ Choose the table or field for which you want to specify permissions.

If you are using a graphical interface, use the Permissions for Selected Table and Permissions for Selected Field radio buttons, then choose the table or field from the selection lists.

If you are using a character interface, use the options at the bottom of the dialog box to choose the table or field.

4 ♦ Specify the security permissions for the table or field. Table 3–1 lists the security permissions you can specify for a table or field.

**Table 3–1: Security Permissions**

| Permission | Description |
|---|---|
| Can-Read | Specifies the users who have permission to read a table. |
| Can-Write | Specifies the users who can write to a table or update records. |
| Can-Create | Specifies the users who can create new records. The user with Can-Create privileges automatically has Can-Write privileges. |
| Can-Delete | Specifies the users who can delete records from a table. |
| Can-Dump | Specifies the users who can dump database or table definitions and data. |
| Can-Load | Specifies the users who can load database or table definitions and data. |

**NOTE:** Use commas, not periods, to separate the names in these options.

Table 3–2 lists the values you use to define the permissions for a table.

**Table 3–2: Access Restrictions for Tables** *(1 of 2)*

| Expression | Meaning |
|---|---|
| * | All users have access. |
| user | This user has access. |

**Table 3–2: Access Restrictions for Tables** *(2 of 2)*

| Expression | Meaning |
|---|---|
| ! user | All users have access, except this user. |
| string* | User IDs that begin with this string have access. |

Any changes you make to the table permissions do not affect the current session or any other current sessions. This means that if other users are working while you change table permissions, they are not affected. To use the new permissions, you must exit and restart Progress.

**NOTE:** Do not try to bypass the Data Dictionary to modify the permissions for the tables and fields in the database. You might lock yourself out of the database.

### Example of Personnel Security

This example shows how to define permissions for a sample database called mywork. Suppose the user IDs defined in the user list for the database are salesrep, inventory, and manager. Users using the IDs are as follows:

- Sales personnel use the salesrep user ID.
- Warehouse personnel use the inventory user ID.
- Managers use the manager user ID.

Suppose you want all users to have Can-Read permission for the customer table, but only users with the specified user IDs to have Can-Write, Can-Create, and Can-Delete privileges. The following example shows how to specify this information for the customer table:

```
  Can-Read: *
 Can-Write: salesrep,manager
Can-Create: salesrep,manager
Can-Delete: manager
```

In the next example for the customer table, all users have Can-Read permission for the Name field. In addition, all users have Can-Write permission, except users with the user ID of inventory:

```
Field name: Name

  Can-Read: *
 Can-Write: *,!inventory
```

In the next example for the customer table, all users, except those with a user ID of inventory, have Can-Read permission for the Max-credit field. Here, only managers have permission to write to this field:

```
Field name: Max-credit

  Can-Read: *,!inventory
 Can-Write: manager
```

### 3.1.3 Determining the Privileges of the Blank User ID

The user has the blank user ID when you run your own startup procedure without using the SETUSERID function, or without using the User ID (-U) and Password (-P) startup parameters. For more information about these startup parameters, see the *Progress Startup Command and Parameter Reference*.

The user can use Progress or a Progress application with a blank user ID and access tables and fields in the database as long as the table- and field-level permissions permit it (blank user ID), and the procedures being run are precompiled. Progress expects an application to run a login program to set the user ID to a more meaningful value.

As the security administrator, you might want to deny privileges to the blank user ID to ensure that unknown users do not gain access to the database. Follow these steps to specify blank user ID privileges:

1 ♦ Access the Data Administration tool if you are using a graphical interface or the Data Dictionary if you are using a character interface.

2 ♦ Choose Admin→ Security→ Disallow Blank Userid Access. Progress prompts you to verify that you want to prevent users with blank user IDs from accessing the working database.

3 ♦ Choose Yes. Progress denies the user all security permissions and inserts an exclamation point (!) at the beginning of all the table and field permissions for the database. You can restore a blank user ID's access to selected tables and fields by modifying the permissions.

## 3.2 Run-time Security

The application developer can provide run-time security to prevent unauthorized users from running precompiled procedures. To establish run-time security, the developer must set up a permissions table within the database.

The permissions table contains records that specify users who are authorized to run specific procedures. Each record in the permissions table must contain at least two fields: an Activity field and a Can-Run field. The Activity field contains the name of the procedure and the Can-Run field contains the user IDs of those who have permission to run the procedure. Within the application, the developer uses the CAN-DO and USERID functions to test whether the current user can run a specific procedure.

As security administrator, you must maintain the permissions table. It is the developer's responsibility to provide the tools to maintain this table.

For more information about run-time security and maintaining the permissions table, see the *Progress Programming Handbook*.

## 3.3 Operating System Security

Each operating system provides security measures that you can use to protect r-code procedure files, procedure library files, and database files.

On UNIX, you can use operating system file security to protect your procedure files in addition to, or instead of, including security checking in the application procedures. For example, if you want to be the only user allowed access to `p-adcust.p`, you should create the source and r-code files with your user ID, then change the permissions for the files. To change the permissions for `p-adcust.p`, enter one of the following UNIX commands:

```
chmod 600 p-adcust.p
chmod 600 p-adcust.r
```

These permissions indicate that you are the only user allowed to read from (R) and (W) write to `p-adcust.p`. No other users are allowed any kind of access to that procedure.

For more information about using the operating system to protect database files, see the *Progress Database Administration Guide and Reference*. For more information about setting operating system permissions, see your operating system documentation.

## 3.4 Designating a Security Administrator

Each level of security uses a user ID to determine the user's authority to access data and files. Associating a user ID with a password provides even more protection against unauthorized access.

If you establish a list of valid user IDs, you must designate one user, or preferably more than one user, as security administrator. The security administrator is responsible for maintaining the security information in the database. Typically, the database administrator also serves as security administrator. The security administrator uses:

- The Data Dictionary to establish connection, schema, and compile-time security
- An activity table provided by the developer to establish run-time security

For more information about designating a security administrator or establishing and maintaining user IDs, passwords, and data access privileges, see the *Progress Database Administration Guide and Reference*.

# 4

# Managing Client Performance

On client systems, Progress consumes system resources to execute Progress 4GL procedures, store variables and record buffers, sort records, and store temporary files. When managing client performance, be aware of key performance factors such as procedure loading and execution, temporary file I/O, and sorting I/O. In addition to these factors, application design and coding can significantly affect client performance. For information about this topic, see the *Progress Programming Handbook*.

This chapter provides information about the following topics:


- "Procedure Loading and Execution"
- "Distributing Progress Files on a Network"
- "Temporary File I/O"
- "Sorting I/O"

## 4.1 Procedure Loading and Execution

*R-code* is the intermediate binary code that Progress generates when it compiles 4GL source files. This is the code that Progress actually runs when it executes a procedure, whether from session compiles or from permanently generated r-code (.r) files. Execution of r-code can affect client performance. This section describes r-code and the performance options related to it. For information about r-code features and functions, see the *Progress Programming Handbook*.

### 4.1.1 Using R-code Libraries to Improve R-code Performance

You can organize and store r-code files in a Progress r-code library. A Progress r-code library is a collection of r-code procedures combined in a single file. R-code libraries allow you to manage and execute r-code procedures more efficiently.

Progress provides two types of r-code libraries: standard and memory-mapped. A standard library contains r-code procedures that execute in local memory. A memory-mapped library contains r-code procedures that execute in shared memory. You create r-code libraries by using the PROLIB utility. For information about using the PROLIB utility to create and manage r-code libraries, see Chapter 6, "Managing R-code Libraries."

When loading and executing r-code procedures from operating system files in a directory, Progress must open and close each file individually. When loading and executing r-code procedures from standard or memory-mapped libraries, Progress opens only one file—the library itself—to access all of the r-code files in the library.

When you execute r-code procedures from either a standard library in local memory or a memory-mapped library in shared memory, you gain the following advantages:

- Faster access to r-code
- Fewer file open and close operations
- Less I/O to temporary files

When you execute r-code procedures from a memory-mapped library in shared memory, you gain the following additional advantages:

- Even faster access to r-code
- Requires less memory because multiple clients access the same r-code segments in shared memory

### 4.1.2 R-code Execution Environment

Progress provides a dynamic environment for loading and executing r-code procedures from operating system files, standard procedure libraries, and memory-mapped procedure libraries. This execution environment consists of the following components:

- **Execution buffer** — The portion of local memory that Progress allocates and uses to store the chain of loaded r-code segments for all standard r-code procedures executing in local memory.
- **Session sort file (.srt)** — A file that Progress uses to dynamically swap standard r-code segments in and out of the execution buffer.
- **Shared memory buffer** — The portion of shared memory that the operating system allocates and uses to store and execute the r-code segments for all memory-mapped r-code procedures.
- **Segment descriptor table** — An in-memory table that references the r-code segments required by all executing procedures, including the location of each r-code segment in local or shared memory and usage count information.
- **R-code directory** — An in-memory table that contains information about all r-code procedures executing in local or shared memory, including r-code size, usage count, segment descriptions, and a reference to the segment descriptor table for each segment.

For more information about the r-code execution environment, see the *Progress Programming Handbook*.

Progress loads and executes r-code procedures in different ways, depending on whether you store the r-code files in an r-code library and the type of library. The following sections provide an overview of how Progress loads and executes r-code from an operating system file, a standard procedure library, and a memory-mapped procedure library.

#### Loading and Executing R-code From a File

When loading an r-code procedure from an operating system file in a directory, Progress opens the r-code file, loads the required r-code segments into local memory, and closes the file.

When executing an r-code procedure from an operating system file, Progress accesses and executes the r-code segments in the execution buffer in local memory. Progress also swaps segments to and from the session sort file, as necessary.

#### Loading and Executing R-code From a Standard Library

When loading an r-code procedure from a standard library, Progress opens the library file and loads the required r-code segments into local memory. The library remains open until the end

of your Progress session or until you remove the library from the PROPATH. When Progress needs to load an r-code segment for a procedure in that library, it accesses the open library in local memory. This is much faster than loading the same procedure from an operating system file.

When executing an r-code procedure from a standard r-code library, Progress accesses and executes the r-code segments in the execution buffer in local memory. Progress does not swap r-code segments to the session sort file unless you specify the PROLIB Swap (-pls) startup parameter. By default, Progress reloads segments from the open library in local memory.

### Loading and Executing R-code From a Memory-Mapped Library

When loading an r-code procedure from a memory-mapped library, Progress opens the library and maps it in shared memory where multiple users can access it. The library remains mapped until the end of your Progress session or until you remove the library from the PROPATH. When Progress needs to execute an r-code segment for a memory-mapped procedure, it accesses the library in shared memory. This is even faster than loading the same procedure from a standard library.

### Tuning R-code Execution

Table 4–1 lists the startup parameters you use to tune the r-code execution environment for a session.

**Table 4–1: Startup Parameters for Tuning the Execution Environment** *(1 of 2)*

| Startup Parameter | Suggested Use |
|---|---|
| Maximum Memory (-mmax) | Use this parameter to set the initial execution buffer ceiling. The default is 3096K. Progress increases the execution buffer dynamically as required, but only after attempting to swap segments to the sort file. To minimize swapping I/O, increase this parameter. |
| Directory Size (-D) | Use this parameter to set the initial number of r-code directory entries. Progress can reuse directory entries for inactive r-code files. However, reusing an entry requires some overhead. To reduce this overhead, increase the value of the -D parameter. The default is 100 entries, the minimum is 5 entries, and the maximum is 500 entries. |

**Table 4–1: Startup Parameters for Tuning the Execution Environment** *(2 of 2)*

| Startup Parameter | Suggested Use |
|---|---|
| Stack size (-s) | When you load data definitions for very large tables or use recursive procedures, you might receive an error message directing you to increase this parameter. This parameter changes the size of the stack (an internal memory area used by Progress program modules). |
| Nested Blocks (-nb) | If memory is severely limited, use this parameter to limit the maximum number of nested procedure blocks allowed. |

For more information about these startup parameters, see the *Progress Startup Command and Parameter Reference*.

### Tuning Standard R-code Libraries

Table 4–2 lists the startup parameters you use to tune standard r-code libraries for a session.

**Table 4–2: Startup Parameters for Tuning R-code Libraries**

| Startup Parameter | Suggested Use |
|---|---|
| PROLIB Memory (-plm) | If you have a large standard library or a memory shortage, use this parameter to allocate a 512-byte cache for the library's directory instead of loading the entire directory into memory. This parameter slows the speed of library access but increases available memory. |
| PROLIB Swap (-pls) | Use this parameter to allow swapping from a standard library to a sort file. Progress typically reads library-stored r-code from the library instead of swapping to a sort file. However, when storing the library over a network, swapping to a local sort file might be faster than accessing the remote library. |

If you specify the PROLIB Memory (-plm) and PROLIB Swap (-pls) startup parameters with memory-mapped libraries, Progress ignores them.

For more information about these startup parameters, see the *Progress Startup Command and Parameter Reference*.

### 4.1.3 Monitoring R-code Activity

You can monitor execution environment activity by using the startup parameters listed in Table 4–3 to collect usage statistics during a Progress client session.

**Table 4–3: Startup Parameters for Monitoring R-code Activity**

| Startup Parameter | Description |
|---|---|
| Statistics (-y) | Collects procedure access and usage statistics throughout the Progress session and writes them to an output file (by default, `client.mon` in your current working directory). Use the SHOW-STATS statement at any time during the session to instruct Progress to write the statistics immediately. |
| Statistics with CTRL-C (-yc) | Collects the same statistics as the -y parameter, but lets you press **CTRL-C** as an alternative to executing the SHOW-STATS statement. Use this parameter if you cannot execute SHOW-STATS. |
| Segment Statistics (-yd) | Collects r-code segment statistics and writes them to the client monitor file (by default, `client.mon` in your current working directory). It provides a breakdown of an r-code file by segment type, including the number of segments and its size. It also summarizes read and write access to the sort file by segment type and access type, including the number of times the sort file was accessed and the number of bytes read from or written to the sort file. |
| Statistics with Cross-reference (-yx) | Collects procedure call statistics and writes them to an output file (by default, `proc.mon` in your current working directory). Use this parameter to answer these questions:<br>• How many calls were made in a given period of time?<br>• How long did a procedure spend executing?<br>• How often was a procedure swapped to and from the sort file? |

For more information about these startup parameters, see the *Progress Startup Command and Parameter Reference*.

### Interpreting R-code Usage Statistics

Figure 4–1 shows a sample excerpt from the output of the Statistics (-y) startup parameter.

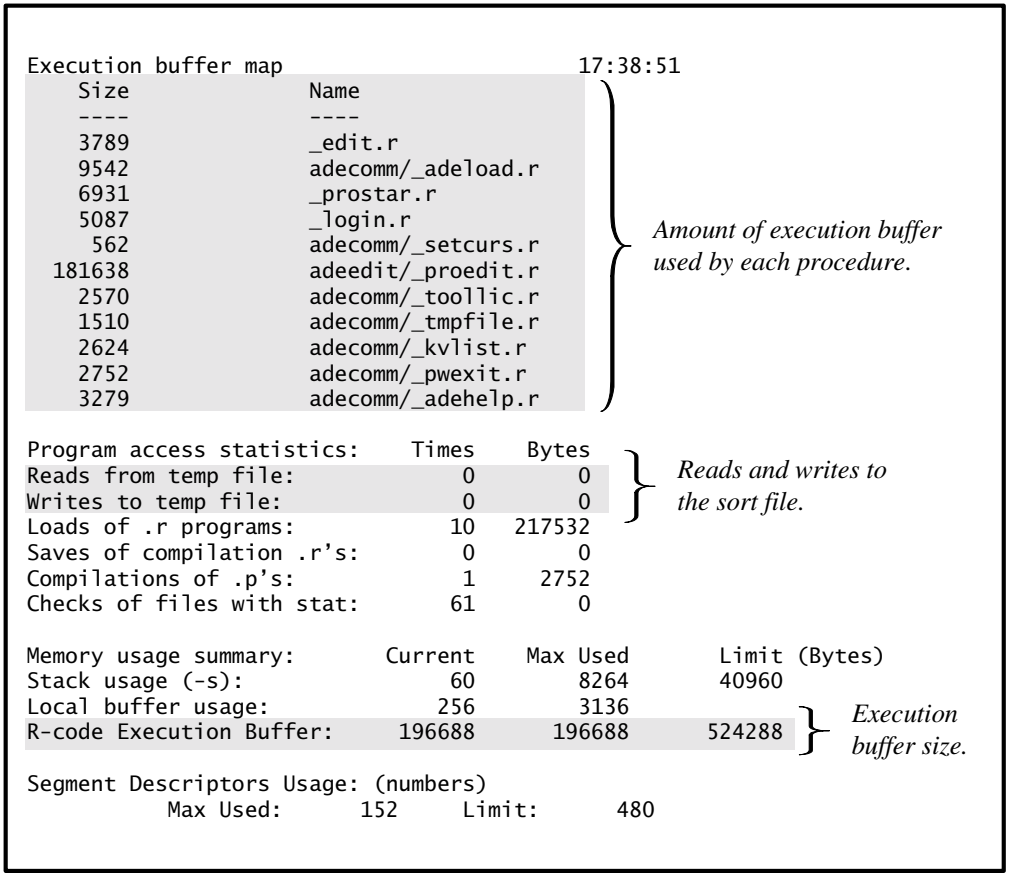



**Figure 4–1: Sample Statistics (-y) Output**

The Reads from temp file and Writes to temp file fields show the amount of swapping to and from the sort file. Segments are written to the sort file only once. Once swapped, they remain in the sort file, even when they are reloaded into memory. For this reason, the number of writes is typically much smaller than the number of reads. If these numbers are relatively high, consider putting the sort files on a dedicated disk. For more information, see the "Temporary File I/O" section later in this chapter.

The R-code Execution Buffer field shows the current size of the execution buffer, the peak usage during the session, and the current ceiling. Use the data in these fields to determine the optimal initial ceiling for the execution buffer (that is, the point at which Progress starts swapping to the sort file). Set this value with the Maximum Memory (-mmax) startup

parameter. For more information, see the "Tuning R-code Execution" section earlier in this chapter.

### Interpreting R-code Segment Statistics

Figure 4–2 shows a sample excerpt from the output of the Segment Statistics (-yd) startup parameter.

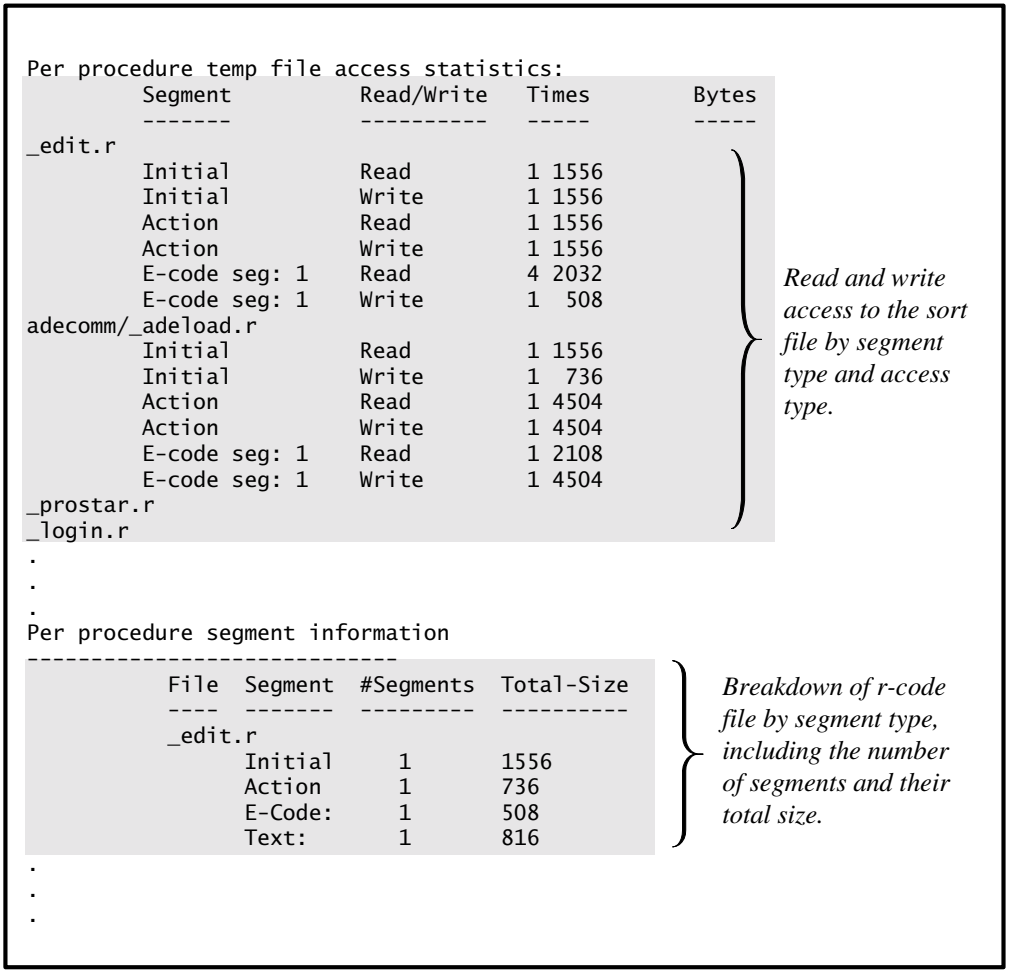



**Figure 4–2: Sample Segment Statistics (-yd) Output**

The Per procedure temp file access statistics fields show the swap rates for each segment of each procedure executed during the session. The Per procedure segment information fields show the number and size of r-code segments in each procedure. Use this information to determine how your application design is affecting r-code performance. For more information about application design and r-code performance, see the *Progress Programming Handbook*.

## 4.2 Distributing Progress Files on a Network

The performance of Progress on your network depends largely on your hardware configuration and where you place the following files:

- Progress system files
- Procedure files
- Temporary files
- Log files

### 4.2.1 Distributing Progress System Files

Progress system software files are the executable and support files that run Progress. The Progress installation procedure places these files in the default `DLC` installation directory on the nodes where you install Progress. If you install Progress on a network file server, you can sometimes improve performance by moving the appropriate system files to the node where they are executed (application workstation or database server machine). This is often true on TCP/IP networks running high-speed UNIX workstations and file servers.

Make sure you copy the correct Progress executables for your LAN protocol. Also, make sure that the operating system on each node knows the new location of the files you copy. Set the appropriate environment variables to point to that directory.

For operating systems with a PATH environment variable or other standard search list convention (such as search directories set with the `map` command in NetWare), be sure that the local Progress installation directory appears before the file server Progress installation directory in the search order. You can do this in the startup file for each network node that receives the Progress system files.

### 4.2.2 Distributing Procedure Files

Procedure files are files that contain Progress source code (.p) or run code (.r). Store procedure files in those directories that Progress can access most rapidly, preferably on a local hard disk attached to each application workstation or Progress AppServer machine.

Once you have distributed your procedure files, modify the PROPATH environment variable in the startup files on each application workstation to include the directory search path that contains those files. The following example sets the PROPATH environment variable to the working directory (.) and the \ACCTPAY directory on the C: drive of a Windows workstation:

```
SET PROPATH=.;C:\ACCTPAY
```

In Windows, you can use the registry or progress.ini file to set this environment variable. For more information, see Chapter 5, "Maintaining User Environments."

### 4.2.3 Distributing Temporary Files

Temporary files are the files that Progress creates during a session and deletes at the end of a session. Because local I/O is usually much faster than network I/O, put temporary files on the local hard disk or a RAM disk on each application workstation, whenever possible. By default, Progress stores temporary files in the user's working directory, whether the working directory is local or remote. To ensure that Progress stores temporary files in a local directory, either start each Progress client in a local directory or with the Temporary Directory (-T) startup parameter set to a local directory.

For more information about temporary files, see the "Temporary File I/O" section later in this chapter. For more information about the Temporary Directory (-T) startup parameter, see the *Progress Startup Command and Parameter Reference*.

### 4.2.4 Distributing Log Files

Progress creates the following log files:

- Event log, which contains a history of significant database events, such as Progress startup and shutdown and system errors. This file always has a .lg extension. Place the event log file in the same directory as the database file.
- Transaction log, if you are using the two-phase commit feature. The log records two-phase commit transactions, and uses a .tl extension. The transaction log file belongs on the same disk as the database.
- Progress AppServer log, if you are using the Progress AppServer option. The log records Progress AppServer broker and server startup and shutdown information, and it contains other system messages that are not directed elsewhere with the OUTPUT TO KEEP-MESSAGES statement. The default location of the AppServer log is the directory where the application broker starts. For information about moving the AppServer log file, see the *Building Distributed Applications Using the Progress AppServer* manual.

## 4.3 Temporary File I/O

Progress creates temporary files during each Progress client session. Each operating system has different temporary files. Table 4–4 describes the temporary files.

**Table 4–4: Temporary Client Session Files**

| File | Description |
|---|---|
| .dbi | Stores temporary tables |
| .lbi | Local before-image file (subtransaction undo) |
| .pge | Edit buffer contents |
| .srt | Temporary sort space; session compile storage |
| .trp | Stores Data Dictionary changes until they are saved |

By default, Progress stores the temporary files in the user's working directory. Use the Temporary Directory (-T) startup parameter to specify an alternate location. When a session ends, Progress deletes these files unless you specify the Save Temp Files (-t) startup parameter. For more information about the Temporary Directory (-T) and Save Temp Files (-t) startup parameters, see the *Progress Startup Command and Parameter Reference*.

On UNIX, unless you use the Save Temp Files (-t) startup parameter, you do not see the temporary files in the directory, because they are created unlinked. However, if your system crashes, the UNIX file system recovery program, `fsck`, finds the files. This program might then prompt you to delete the files. If this occurs, delete them.

On Windows, these temporary files are always visible during a Progress session, but the file sizes are set to zero. Windows does not record file sizes until files are closed.

Writing to these temporary files creates I/O activity that might slow down client performance. The amount of temporary file I/O activity varies among applications.

You might find that running a particular procedure results in Progress abnormally terminating in a way that indicates the hard disk is full. However, when you check the disk immediately afterward, adequate space is available. One possibility is that temporary files created during a Progress session become quite large, taking up disk space as they expand. Since Progress erases temporary files at the end of a session, they do not appear when you check the disk. Of the temporary files, the following files are most likely to cause this problem:

- Temp table file (.dbi) — Progress uses this file to store temporary tables. If the application uses a lot of temporary tables, this file can grow quite large.

- Local before-image file (.lbi) — Progress uses this file to back out of subtransactions and variable value changes. If the local before-image file becomes too large, the procedures probably use a lot of subtransactions. Examine the transaction structure of the application procedures to understand why Progress starts so many subtransactions. For more information about transaction processing, see the *Progress Programming Handbook*.

- Session sort file (.srt) — Progress uses this file to dynamically swap r-code segments in and out of the execution buffer. Use the r-code usage statistics startup parameters to collect information about I/O to the sort file, as explained in the "Monitoring R-code Activity" section earlier in this chapter. Using r-code libraries helps reduce the amount of I/O required to load r-code files. For more information, see the "Using R-code Libraries to Improve R-code Performance" section earlier in this chapter.

If temporary files grow to exceed disk space, change the application to reduce the size of temporary files, or use the Temporary Directory (-T) parameter to store the temporary files on a disk with sufficient space to hold the files. To optimize performance, place the temporary files on a dedicated disk.

## 4.4 Sorting I/O

Table 4–5 lists the startup parameters you use to improve performance when sorting records for reports and during index rebuild operations.

**Table 4–5: Startup Parameters for Tuning Sort Performance**

| Startup Parameter | Description |
|---|---|
| Speed Sort (-TB) | Increase this parameter to 8K, 16K, or even 31K, particularly for index rebuild operations. Increasing this parameter increases memory usage during sorting and index rebuilds. It also slightly increases disk space usage. |
| Merge Number (-TM) | Increase this parameter to 32 to speed the merge phase of the sort process (at the cost of increased temporary memory usage). |

For more information about these startup parameters, see the *Progress Startup Command and Parameter Reference*.

# 5

# Maintaining User Environments

This chapter explains how to maintain user interface environments in Windows and on UNIX.


- “Maintaining the Windows User Environment”
- “Maintaining the UNIX User Environment”

## 5.1 Maintaining the Windows User Environment

In the Windows software environment, Progress uses the `progress.ini` file as well as the Registry to maintain environment variables.

When you install Progress, the installation program automatically updates the Registry with the information in the `progress.ini` file that is shipped to you. Progress Software recommends that you maintain environment variables in both the `progress.ini` file and the Registry. If you modify environment variables in the `progress.ini` file, you must update the information in the Registry.

Although there are several methods for adding and deleting information from the Registry directly, Progress Software recommends that you do not use these direct update methods. Instead, maintain the Registry information by editing the `progress.ini` file and using the INI2REG utility to update the Registry. This approach synchronizes the information in the `progress.ini` file and the Registry without causing unintended edits to the Registry.

For more information about modifying the `progress.ini` file, see the "Maintaining the progress.ini File" section later in this chapter.

### 5.1.1 Using the INI2REG Utility

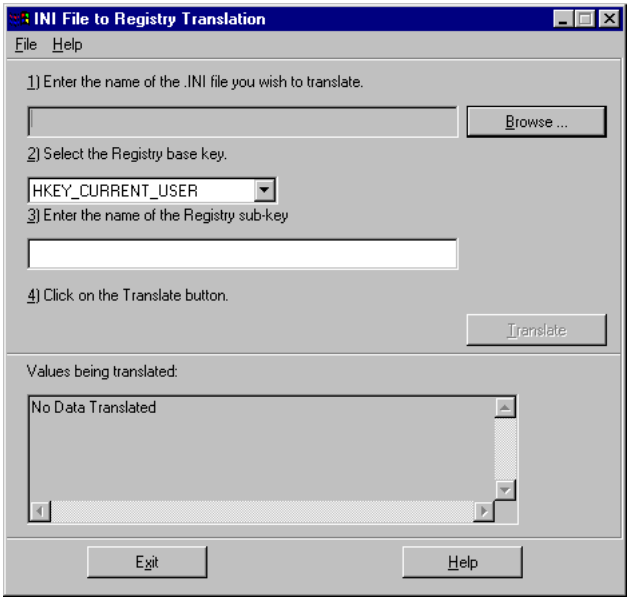
The INI2REG utility lets you translate the contents of an initialization (.ini) file into comparable Registry key entries. You can access the INI2REG utility through a graphical user interface or from the command line.

#### Using the Graphical User Interface

To access the INI2REG utility through its graphical user interface, specify the INI2REG command using the following syntax:

```
ini2reg
```

The INI File to Registry Translation dialog box appears:




To translate an initialization file:

1 ♦ From the INI File to Registry Translation dialog box, choose the Browse button or File→ Open. The Open dialog box appears.

2 ♦ Type in or browse for the initialization file you want to translate, and choose OK. The name of the selected file appears in the .INI file to translate field.

3 ♦ From the Registry base key drop-down list, select the base key under which you want to create the subkey entries. The default subkey that corresponds to the selected initialization file and base key appears in the Registry subkey field.

If you want to use a subkey other than the default, type the subkey name in the Registry subkey field.

4 ♦ To begin the translation, choose the Translate button or File→ Translate. For each entry in the initialization file, INI2REG creates a Registry key entry with the same value. As each entry is translated, it appears in the Values being translated display area. If a Registry key entry already exists, INI2REG prompts you to overwrite it.

5 ♦ To exit, choose the Exit button or File→ Exit.

### Using the Command Line Interface

To access the INI2REG utility and translate an initialization file from the command line, specify the INI2REG command using the following syntax:

```
ini2reg { -a | -ao } [ -i ini-file-name ] [ -b base-key ]
   [ -s subkey ] [ -d default-subkey ]
```

-a

Automates the translation. That is, it processes the translation automatically without presenting the graphical user interface.

-ao

Automates the translation and automatically overwrites existing Registry entries.

-i *ini-file-name*

The pathname of the initialization file to translate.

-b *base-key*

The base key under which the subkey is created. The value of *base-key* must be one of following:

- HKEY_CURRENT_USER (default)
- HKEY_USERS
- HKEY_LOCAL_MACHINE
- HKEY_CLASSES_ROOT
- HKEY_CURRENT_CONFIG
- HKEY_DYN_DATA

-s *subkey*

The subkey under which entries are created. The subkey is created under the base key specified in the -b parameter.

Use -d to specify a default subkey for the translation destination.

-d *default-subkey*

The default subkey under which entries are created. The default subkey is created under the base key specified in the -b parameter.

If the base key is HKEY_LOCAL_MACHINE or HKEY_CURRENT_USER, INI2REG prepends "SOFTWARE" to the specified default subkey. If the initialization file name is not `progress.ini`, INI2REG appends the root name of the initialization file to the specified default subkey. If this parameter is not specified, INI2REG uses a default subkey of `PSC\PROGRESS\`*`version`*.

You can generalize this as follows:

```
base-key\[SOFTWARE]\default-subkey\[rootname]
```

Use -s to specify a complete subkey for the translation destination.

By default, the INI2REG utility creates Registry key entries at the following location:

```
HKEY_CURRENT_USER\SOFTWARE\PSC\PROGRESS\version
```

### 5.1.2 Searching progress.ini and the Registry at Startup

At startup, Progress searches for a Registry location, a `progress.ini` file, or both. The particular search path depends upon the parameters you specify with the `prowin32` command, particularly the Registry Base Key (-basekey) and Initialization File (-ininame) parameters. You can use these parameters to create a more efficient search at startup.

**NOTE:** The -basekey and -ininame parameters can only be used on the command line. You cannot use them within a parameter file.

Figure 5–1 illustrates the path that Progress follows when searching for environment information in Windows.

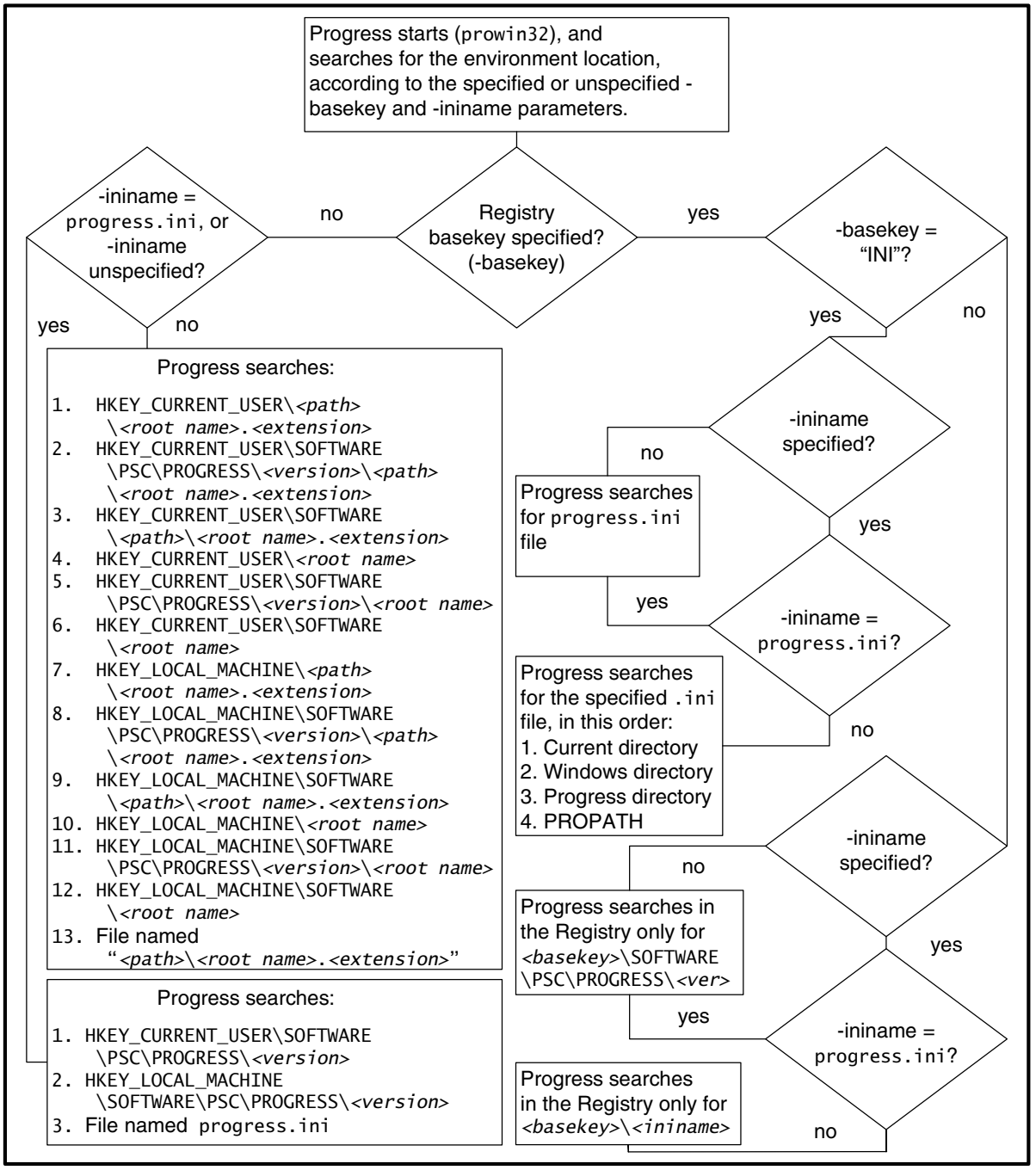



**Figure 5–1: Windows Environment Information Search Path**

When Progress does not find environment information in the Registry, or when it was started with the -basekey INI startup parameter, it searches through directories for a `progress.ini` file in the following order:

1. The current working directory (as defined in the Working Directory field in the Program Item properties sheet for Windows).
2. The `WINDOWS` directory.
3. The directory that contains the Progress executable.
4. The user's PROPATH.

This search order fosters deployment flexibility because you can:

- Create a customized `progress.ini` file.
- Place the file in a consistent place on every machine.
- Allow all users to share the same file on a network.

After Progress locates the environment information, Progress reads the values for DLC, PROMSGS, and other environment variables. For each variable, Progress first searches the location where it found the environment information. If Progress does not find the variable, it searches for an operating system environment variable.

Progress expects to find all environment settings in the same location it finds the environment variables. There is one exception: Progress searches HKEY_CURRENT_USER and HKEY_LOCAL_MACHINE first when it starts without the -basekey parameter and finds the environment information in the Registry. If Progress writes a value to the Registry, however, it will always write it to HKEY_CURRENT_USER. This allows multiple users of a single machine to customize settings individually. The environment setting location could be either a Registry key or an initialization file, but not both.

The Progress 4GL GET-KEY-VALUE statement also searches for user environment information. The GET-KEY-VALUE statement searches either the Registry or a `progress.ini` file, but not both. When GET-KEY-VALUE searches the Registry, it searches both of the HKEY_CURRENT_USER and HKEY_LOCAL_MACHINE keys if the following conditions are true:

- The Progress startup command (prowin32) used for the current session did not include the -basekey parameter.
- Progress found the environment information in the Registry at startup or at the last LOAD statement.

The Progress 4GL PUT-KEY-VALUE statement writes to the location where the environment information was found at startup. If that location is the Registry, and if the prowin32 command did not include the -basekey parameter at startup, the PUT-KEY-VALUE statement writes to the HKEY_CURRENT_USER key and not to the HKEY_LOCAL_MACHINE key.

### 5.1.3 Maintaining the progress.ini File

The `progress.ini` file specifies environment variable settings for the Windows environment. It contains sections and settings for the following types of options:

- Options required by Progress
- Options required by Progress tools (such as the Procedure Editor and the Report Builder)
- User-defined options
- Environment variables

Figure 5–2 shows excerpts from a sample `progress.ini` file.

```
[Startup]
V6Display=NO
V6FontNumber=3
DefaultV6UpdateFont=fixedsys,underline
ImmediateDisplay=NO
MultitaskingInterval=0
DefaultFont=system
DefaultFixedFont=fixedsys
PrinterFont=courier new, size=10
PrinterFont1=courier new, size=8
PrinterFont2=courier new, size=12
   .
   .
   .
[WinChar Startup]
PROPATH=.,C:\DLC
PROMSGS=C:\DLC\PROLANG\PROMSGS.GER
PROCFG=C:\DLC\PROGRESS.CFG
PROBUILD=C:\DLC\PROBUILD
DLC=C:\DLC
EnableMouse=YES
V6Keys=NO
V6FKeys=NO
   .
   .
   .
```

**Figure 5–2: Sample Progress Initialization File in Windows**

When installed, the `progress.ini` file contains the following sections:

- "Startup Section"
- "Colors Section"
- "Default Window Section"
- "Fonts Section"
- "WinChar Startup Section"
- "WinChar Colors Section"
- "WinChar Default Window Section"
- "WinChar Keys Section"

The `progress.ini` file can also contain the following optional sections:

- "Debug-Init Section"
- "Debug-Macros Section"
- "Debug-Buttons Section"
- "Keys Section"
- "Keymap Section"

You can edit the `progress.ini` file to customize environments at different sites. The `progress.ini` file is located in the `$DLC/probuild/eucapp` directory, by default.

The following sections describe each of these `progress.ini` file sections.

### Startup Section

The Startup section of the `progress.ini` file contains options that specify the following environment characteristics for graphical user interface clients:

- Dynamic link libraries used for AS/400 communications
- Display attributes and multi-tasking capabilities
- Default fonts used for displaying and printing alphanumeric data

The Startup section also contains Progress environment variables that you can set for use with graphical user interface clients. See Table 5–5 in the "Specifying Environment Variables" section later in this chapter for a list of environment variables you can set in the Startup section.

The Startup section contains the following options:

`AlignFrameTitles`

```
AlignFrameTitles=n
```

Lets you specify the alignment of frame titles in a graphical user interface. Valid settings include the following: 0 (left justified), 1 (centered), and 2 (right justified). The default is centered.

`AS/400_AppcDllName`

```
AS/400_AppcDllName=dll-filename
```

Lets you specify a dynamic link library for AS/400 communications.

For Version 7.3B and later, you can specify the name of the DLL that you want to use for AS/400 communications. Table 5–1 lists the available DLLs depending on the AS/400 communication protocol and the type of network support you use.

**Table 5–1: DLL Files for AS/400 Communications**

| AS/400 Communication Protocol | Type of Network Support | DLL Filename |
|---|---|---|
| SNA | Rumba for APPC | AS4WCP32.DLL |
| SNA | Client Access/Netsoft | AS4EHN32.DLL |
| SNA | Microsoft SNA Server | AS4WCP32.DLL |

CarefulPaint

```
CarefulPaint={ YES | NO }
```

Lets you set the accuracy with which Progress repaints overlapping 3D widgets and frames. Set to YES for Progress to repaint overlapping 3D widgets and frames slowly and carefully. Set to NO for Progress to repaint overlapping 3D widgets and frames quickly (which might reduce the quality of the graphic). The default value is YES.

This option applies only when UseNative3D=NO.

ClientTimeOut

```
ClientTimeOut=n
```

Lets you specify the number of minutes a client session can remain inactive before the server disconnects the session.

The server process determines whether a client has been inactive for a specified period of time, and if so, the server disconnects the client and backs out any related active transactions.

DefaultFixedFont

```
DefaultFixedFont=typeface, [ size point-size, keyword, ... ],
   [ script=scriptname ]
```

Lets you specify the default fixed display font.

*typeface*

A valid typeface name, such as Times or Courier.

*point-size*

The size of the font, in points. If you omit the point size, Progress uses the default point size as defined by the typeface.

*keyword*

You can specify the following keywords: bold, italic, underline, or strikeout.

*scriptname*

By default, Progress uses the script (character set) of the Windows FixedSys font for all fonts specified in the environment. If you select a different script when editing a font using the font common dialog, Progress writes the specified script name to the font entry.

DefaultFont

```
DefaultFont=typeface, [ size point-size, keyword, ... ],
   [ script=scriptname ]
```

Lets you specify the default display font.

*typeface*

A valid typeface name, such as Times or Courier.

*point-size*

The size of the font, in points. If you omit the point size, Progress uses the default point size as defined by the typeface.

*keyword*

You can specify the following keywords: bold, italic, underline, or strikeout.

*scriptname*

By default, Progress uses the script (character set) of the Windows FixedSys font for all fonts specified in the environment. If you select a different script when editing a font using the font common dialog, Progress writes the specified script name to the font entry.

DefaultV6UpdateFont

```
DefaultV6UpdateFont=typeface, [ size point-size, keyword, ... ],
   [ script=scriptname ]
```

Lets you specify the Version 6 update font.

*typeface*

A valid typeface name, such as Times or Courier.

*point-size*

The size of the font, in points. If you omit the point size, Progress uses the default point size as defined by the typeface.

*keyword*

You can specify the following keywords: bold, italic, underline, or strikeout.

*scriptname*

By default, Progress uses the script (character set) of the Windows FixedSys font for all fonts specified in the environment. If you select a different script when editing a font using the font common dialog, Progress writes the specified script name to the font entry.

When V6Display=YES, the font specified by the DefaultV6UpdateFont setting is used when a fill-in is in update mode. The default value for DefaultV6UpdateFont is a version of the system default fixed font with underlines (`fixedsys`, `underline`).

The DefaultFixedFont and the DefaultV6UpdateFont should be the same base font. If you change them, make sure that the font is fixed and that the DefaultV6UpdateFont has the underline attribute. Progress does not automatically add the underline attribute to the DefaultV6UpdateFont setting.

`EndMsgTitle`
`EndMsgText`

```
EndMsgTitle=title-text
EndMsgText=message-text
```

Lets you specify the title and text for the Windows shutdown message box.

When Windows shuts down, a dialog box prompts the user to verify the shutdown. If the user answers YES, Progress closes and the Windows Exit procedure continues. If the user answers NO, the Windows Exit procedure terminates and Windows remains active.

EndMsgTitle can be up to 80 characters long. Its default value is “Progress”. EndMsgText can be up to 80 characters long. Its default value is “Windows is exiting. Is this OK?”.

FrameSpacing

```
FrameSpacing=n
```

Lets you specify how the Progress layout processor spaces frames.

Setting the value to -1 places the frame at the next character unit boundary. Setting the value to 0 places frames one half of a character unit apart. Setting the value greater than 0 spaces the frames by the specified number of pixels.

ImmediateDisplay

```
ImmediateDisplay={ YES | NO }
```

Lets you redisplay the screen at predefined intervals.

If you do not set this option, some Version 6 applications will not draw the screen correctly. Set this option for applications with existing code that you do not want to modify. The default setting is NO.

You can also use the IMMEDIATE-DISPLAY attribute with the SESSION system handle to surround the offending code. This is the preferred technique.

For more information about the IMMEDIATE-DISPLAY attribute and the SESSION system handle, see the *Progress Language Reference*.

Keep3DfillinBorder

```
Keep3DfillinBorder={ YES | NO }
```

Set to YES to display a disabled fill-in field with a border. Set to NO to display a disabled fill-in field without a border. The default value is NO.

MultitaskingInterval

```
MultitaskingInterval=n
```

Lets you specify how the Progress session interacts with a Windows cooperative multi-tasking environment. Its value determines how often Progress filters events or

messages between itself and other Windows applications. As Progress filters these events more often, it executes procedures less efficiently but allows other Windows applications more opportunity to execute. Adjusting the internal event filter is particularly useful during background processing, such as report generation.

To maximize performance during batch-mode processing, set the value to 9999 (the maximum value allowed). If you want to run another application while you run Progress in batch mode, set the value to 1000. If you set the value to 0 (the default), the internal loop never executes. Although this results in high performance, interoperability with other Windows applications is poor. The lower the number, the more often the loop executes, resulting in better interoperability. Set this option for applications with existing code that you do not want to modify.

You can also use the MULTITASKING-INTERVAL attribute with the SESSION system handle to control how Progress interacts with Windows cooperative multi-tasking by surrounding long 4GL processing loops with this attribute.

See the *Progress Language Reference* for more information about the MULTITASKING-INTERVAL attribute and the SESSION system handle.

`Opsys`

```
Opsys=character-string
```

Lets you specify the value used by the OPSYS preprocessor directive or the value returned by the OPSYS function. Valid settings include any character string.

`PopupAppOverridesSys`

```
PopupAppOverridesSys={ YES | NO }
```

Set to YES to display the system-default pop-up menu only if a user-defined pop-up menu is not available. Set to NO to display the system-default pop-up menu regardless of a user-defined pop-up menu. If a user-defined pop-up menu is available, it is displayed after the system pop-up menu is closed. The default value is YES.

**NOTE:** The system-default pop-up menu is displayed when you click the right mouse button on a fill-in field or an editor widget and a user-defined pop-up menu is not available.

PopupNoSysDefault

```
PopupNoSysDefault={ YES | NO }
```

Set to YES to suppress the system-default pop-up menu. Only the user-defined pop-up menu is displayed. The default value is NO.

**NOTE:** The system-default pop-up menu is displayed when you click the right mouse button on a fill-in field or an editor widget and a user-defined pop-up menu is not available.

PopupOnMouseDown

```
PopupOnMouseDown={ YES | NO }
```

Set to YES to display a pop-up menu when the right mouse button is depressed. Set to NO to display a pop-up menu when the right mouse button is released. The default value is NO.

**NOTE:** This option applies only to pop-up menus that are displayed with the right mouse button (that is, MENU-MOUSE = 3). Menus that are displayed with the left or middle mouse button are always displayed when the buttons are depressed.

PrinterFont

```
PrinterFont=typeface, [ size=screen-point-size ]
```

Lets you specify the printer fonts Progress uses with the OUTPUT TO PRINTER statement.

PrinterFont*n*

```
PrinterFontn=typeface, [ size=screen-point-size ]
```

Lets you specify the printer font Progress uses with the OUTPUT TO LPT*n* statement.

UseNative3D

```
UseNative3D={ YES | NO }
```

Set to YES to use the 3D drawing capabilities in Windows 95/NT 4.0. Set to NO to use the 3D drawing capabilities in Progress. The default value is YES.

Use-3D-Size

```
Use-3D-Size={ YES | NO }
```

Lets you specify how Progress calculates the height of a character unit.

If you set the value to YES, Progress uses the height of a three-dimensional fill-in. If you set the value to NO, Progress uses the height of a non-three-dimensional fill-in. The default setting is YES.

V6Display

```
V6Display={ YES | NO }
```

Lets you specify Progress Version 6 behavior.

When you compile a procedure in Windows, Progress compiles the borders and fonts associated with user interface widgets into the screen layout by default. As a result, you might encounter compilation problems when you try to run a Version 6 application in Progress Version 7 for Windows. There might not be enough space on the display to accommodate the borders and default fonts that are automatically associated with user interface widgets of the Version 6 application.

To avoid this problem, set V6Display to YES to specify that you are running Version 6 code.

You can also set a session attribute to control the run-time setting of the V6Display state. The attribute is V6DISPLAY and has a Boolean value. Setting this attribute has the same effect as specifying the setting at startup. However, you must hide all existing windows.

When you set the V6Display option or session attribute:

- The character unit height is recalculated to the height of the font specified by the DefaultV6UpdateFont option.
- The character unit width is recalculated to the width of an average character in the font specified by the DefaultV6UpdateFont option.
- The default font is automatically changed to use the font specified by the DefaultFixedFont option.
- All Progress fill-ins are created without a border.
- Progress fill-ins keep any trailing spaces.
- All Progress fill-ins that are enabled for update use the font specified by the DefaultV6UpdateFont option (by default, the default fixed font with underlines). This means that updateable fill-ins are drawn with underlines.
- Frame titles are drawn so that the frame title height measures exactly one character unit.

The V6Display option or attribute does not enable support for PUT SCREEN. In addition, if you set the V6Display option, Version 7 applications that have not been compiled with the V6Display option set to YES will draw screens incorrectly.

**NOTE:** The V6Display option is intended for use with applications, not with the Progress ADE. The ADE is not supported when V6Display is set to YES.

If you plan to use Version 6 user interface code with Version 7 tools (or your own Version 7 interface), Progress Software recommends that you use the V6Frame option instead of V6Display.

`V6FontNumber`

```
V6FontNumber=n
```

Lets you specify the font for Version 6 frames.

By default, this option points to the font defined as font3 in the [Fonts] section. Font3 is the ADE character-mode font. If you change the V6FontNumber value, make sure you define the new font number in the [Fonts] section.

**NOTE:** Do not change the definition of font3 in the [Fonts] section. Progress relies on that font definition for other purposes.

`V6FrameBGC`

```
V6FrameBGC=n
```

Lets you specify the default background color for all frames that use the V6FRAME option. Valid settings include any number that corresponds to a color in the Progress color table.

`WindowSystem`

```
WindowSystem=character-string
```

Lets you specify the value used by the WINDOW-SYSTEM preprocessor directive. Valid settings include any character string.

### Colors Section

The Colors section of the `progress.ini` file contains options that specify the colors that make up the color table for use with graphical user interface clients. There are 16 colors available to an application for use as either foreground or background colors. You can specify up to 256 colors.

**NOTE:** Do not change the first 16 colors. These colors are reserved for the Application Development Environment (ADE) tools. Changing them could cause the Progress ADE tools to malfunction.

Each entry in the Colors section performs two semantic functions: it defines a color, and it maps that color to an integer in the range 0 to 255. An application specifies this integer when making a color assignment to a widget.

Specify a color entry using the following syntax:

```
colorn = { R , G , B , | colorname }
```

*n*

An integer from 0 to 255 that specifies the color table entry.

R

An integer that specifies the amount of red present in the color.

G

An integer that specifies the amount of green present in the color.

B

An integer that specifies the amount of blue present in the color.

*colorname*

Any of the following color names, mapped to the colors defined in the Windows Control Panel:

```
COLOR-SCROLLBAR
COLOR-BACKGROUND
COLOR-ACTIVECAPTION
COLOR-INACTIVECAPTION
COLOR-INACTIVECAPTIONTEXT
COLOR-MENU
COLOR-WINDOW
COLOR-WINDOWFRAME
COLOR-MENUTEXT
COLOR-WINDOWTEXT
COLOR-CAPTIONTEXT
COLOR-ACTIVEBORDER
COLOR-INACTIVEBORDER
COLOR-APPWORKSPACE
COLOR-HIGHLIGHT
COLOR-HIGHLIGHTTEXT
COLOR-BTNFACE
COLOR-BTNHIGHLIGHT
COLOR-BTNSHADOW
COLOR-GRAYTEXT
COLOR-BTNTEXT
```

Any entries you add to the color table must be sequential. For example, since the installed `progress.ini` file defines color0 to color15, the next color you add must be color16. If you add an entry for color17, Progress ignores it if color16 is undefined.

For backward compatibility, color0 to color15, as installed, are the same as the colors supported in Version 6. See the chapter about colors and fonts in the *Progress Programming Handbook* for a listing of these colors.

For backward compatibility, the progress.ini file specifies the following color pairs for Version 6 applications:

- NORMAL specifies the colors for fill-ins that do not have input focus.
- INPUT specifies the colors for fill-ins that have input focus.
- MESSAGES specifies the colors for the message area.

NORMAL, INPUT, and MESSAGES each take a pair of integer values from 0 to 255. The integers represent color table entries. For example, in the default progress.ini file, NORMAL is defined as foreground equal to color table entry 0 (RGB value 0,0,0) and background equal to color table entry 15 (RGB value 255,255,255).

The following example shows how to redefine NORMAL as foreground equal to color table entry 3 and background equal to color table entry 12:

```
NORMAL=3,12
```

### Default Window Section

The Default Window section of the progress.ini file specifies the following options for a graphical user interface client's default window:

- The location; specify the location as an x,y pixel value.
- The size; specify the size as a number of rows and columns. The default size is 25 by 80.

Following is a sample Default Window section:

```
[Default Window]
x=
y=
rows=
columns=
```

### Fonts Section

The progress.ini file as installed contains font table entries for the Progress ADE tools. Progress uses the default system fonts defined in the Startup section, unless you add additional fonts to the existing font table in the Fonts section.

**NOTE:** You can add site-specific fonts to the file. Do not change the first 8 fonts. These fonts are reserved for the Application Development Environment (ADE) tools. Changing them could cause the Progress ADE tools to malfunction.

Determine which screen fonts exist in your environment. The fonts you choose should be graphically compatible. One way to choose fonts is to use the font common dialog box.

The font table can contain up to 256 entries. Each entry in the table performs two semantic functions: it names a font that exists in your environment and it maps that font to an integer from 0 to 255. An application specifies this integer when making a font assignment to a widget.

**NOTE:** You cannot enter extra spaces in the font definition section. Extra spaces will cause the error 4499.

Specify a font entry using the following syntax:

```
fontn=typeface, [ size point-size, keyword, ... ],
   [ script=scriptname ]
```

*n*

An integer that specifies the font table entry.

*typeface*

A valid typeface name, such as Times or Courier.

*point-size*

The size of the font, in points. If you omit the point size, Progress uses the default point size as defined by the typeface.

*keyword*

You can specify the following keywords: bold, italic, underline, or strikeout.

*scriptname*

By default, Progress uses the script (character set) of the Windows FixedSys font for all fonts specified in the environment. If you select a different script when editing a font using the font common dialog, Progress writes the specified script name to the font entry.

### WinChar Startup Section

The WinChar Startup section of the `progress.ini` file contains options that specify the following environment characteristics for character clients:

- Default interface attributes
- Default key bindings

The WinChar Startup section also contains Progress environment variables that you can set for use with character clients. See Table 5–5 in the "Specifying Environment Variables" section later in this chapter for a list of environment variables you can set in the WinChar Startup section.

The WinChar Startup section contains the following options:

`BrowseRowMarker`

```
BrowseRowMarker=n
```

Lets you specify a character to represent a browse row marker. Valid settings include any number that corresponds to a character in the CPSTREAM character set.

`CursorHeight`

```
CursorHeight=n
```

Lets you specify the height of a cursor as a percentage. Valid settings include any number between 0 and 100. This option overrides the UseDosCursor setting.

`EnableMouse`

```
EnableMouse={ YES | NO }
```

Set to YES to enable mouse input. Set to NO to disable mouse input. The default value is YES.

`MenuMnemonicColor`

```
MenuMnemonicColor=n
```

Lets you specify the color of mnemonic characters in menu items. Valid settings include any number that corresponds to a color in the Progress color table.

SingleLineBorder

```
SingleLineBorder=n,n,n,n,n,n,n,n
```

Lets you specify the characters used to draw borders and lines (such as frame borders, rectangles, submenu and pop-up menu borders, and horizontal and vertical lines). Valid settings include any eight numbers that correspond to characters in the CPSTREAM character set, separated by commas.

These eight numbers represent the corners and sides of a rectangle and must be specified in the following order: top, bottom, left, right, top-left corner, top-right corner, bottom-right corner, and bottom-left corner.

SysCheckmark

```
SysCheckmark=n
```

Lets you specify a character to represent a selection marker in widgets (such as a selection list, menu, or browser). Valid settings include any number that corresponds to a character in the CPSTREAM character set.

UseDosCursor

```
UseDosCursor={ YES | NO }
```

Set to YES to change the cursor from an underscore to a solid block when switching from insert mode to over-strike mode in a fill-in field or editor widget. Set to NO to use an underscore as the cursor in either mode. The default value is NO.

V6Keys

```
V6Keys={ YES | NO }
```

Set to YES to use a set of key bindings that are Version 6 compatible by default. The default value is NO.

V6FKeys

```
V6FKeys={ YES | NO }
```

Set to YES to use the bindings for the **CTRL**, **ALT**, **SHIFT**, and **F13** to **F39** keys that are Version 6 compatible by default. The default value is NO.

### WinChar Colors Section

The WinChar Colors section of the `progress.ini` file contains options that specify the colors that make up the color table for use with character clients.

Each entry in the WinChar Colors section performs two semantic functions: it defines a color, and it maps that color to an entry in the color table by specifying an integer from 0 to 255. An application specifies this integer when making a color assignment to a widget.

Specify a color entry using the following syntax:

```
keyword | colorn = { [ BLINK- ] [ BRIGHT- ] fgnd-color | bgndcolor }
```

*keyword* | color*n*

You can also specify color0, color1, and color2 using the NORMAL, INPUT, and MESSAGES keywords, respectively. The keyword overrides the color*n* specification. That is, if you specify both color*n* and its corresponding keyword, Progress displays the colors you specified with the keyword.

*fgnd-color*

The color to use for the screen foreground.

*bgnd-color*

The color to use for the screen background.

For information about color definitions for the WinChar Colors section, see the Color Phrase reference entry in the *Progress Language Reference*.

### WinChar Default Window Section

The WinChar Default Window section of the `progress.ini` file specifies the line mode for a character client. You can specify 50 rows if your hardware supports 50-line mode. Specify 25

rows if you want the application to be compatible with Progress Version 6 or if you want to port the application to a character terminal.

The following is an example WinChar Default Window section:

```
[WinChar Default Window]
rows=25
```

### WinChar Keys Section

The WinChar Keys section of the `progress.ini` file defines the customized key bindings for use with character clients. Table 5–2 lists the user-definable key bindings in the Windows environment. For a complete listing of the Windows key function and key label mappings for handling user input, see the *Progress Programming Handbook*.

**Table 5–2: User-definable Key Bindings in Windows** *(1 of 2)*

| Key Function | Key Label |
|---|---|
| APPEND-LINE | CTRL-A |
| BLOCK | CTRL-V |
| BREAK-LINE | CTRL-ALT-B |
| CLEAR | F8 |
| COPY | F11 |
| CUT | F10 |
| DEFAULT-POP-UP | SHIFT-F4 |
| DELETE-LINE | CTRL-D |
| END-ERROR | ESC |
| ENTER-MENUBAR | F3 |
| FIND | CTRL-F |
| GET | F5 |
| GO | F1 |

**Table 5–2: User-definable Key Bindings in Windows** *(2 of 2)*

| Key Function | Key Label |
|---|---|
| HELP | F2 |
| INSERT-MODE | INS |
| NEW-LINE | CTRL-N |
| NEXT-FRAME | CTRL-TAB |
| PASTE | F12 |
| PREV-FRAME | CTRL-SHIFT-TAB |
| PUT | F6 |
| RECALL | F7 |

Specify a key-binding entry using the following syntax:

```
keyfunction=keylabel [ , keylabel ]
```

Use a single entry to specify a key binding; additional entries for the same key function are ignored.

If you specify more than one key label, Progress uses the first when it needs to display a key label given a key function, as in "Enter data or press Escape to end."

For example, the following entry maps the Help key function to the F2 key label, and the Go key function to the F1 key label:

```
HELP=F2
GO=F1
```

You can use any combination of **CTRL**, **SHIFT**, and **ALT** keys. However, **SHIFT** only has an effect when used with ASCII keys. If an entry contains nonexistent key labels, the entry is ignored.

**NOTE:** Set the V6Keys option in the [WinChar Startup] section to enable Progress to use a set of key bindings that are Version 6 compatible by default.

### Debug-Init Section

Do not modify information in the Debug-Init section.

### Debug-Macros Section

The Debug-Macros section of the `progress.ini` file contains a list of parameters that define initial macros used by the Debugger. You can also define your own macros. For more information, see the *Progress Debugger Guide*.

### Debug-Buttons Section

The Debug-Buttons section of the `progress.ini` file contains a list of parameters that define initial buttons used by the Debugger. You can also define your own buttons. For more information, see the *Progress Debugger Guide*.

### Keys Section

The Keys section is an optional section you can create in the `progress.ini` file to customize key bindings for the Windows environment.

Table 5–3 lists the static key bindings in the Windows environment.

**Table 5–3: Static Key Bindings in Windows** *(1 of 2)*

| Key Function | Key Label |
|---|---|
| BACKSPACE | BACKSPACE |
| BACK-TAB | SHIFT-TAB |
| CURSOR-DOWN | CURSOR-DOWN |
| CURSOR-LEFT | CURSOR-LEFT |
| CURSOR-RIGHT | CURSOR-RIGHT |
| CURSOR-UP | CURSOR-UP |
| DELETE-CHARACTER | DEL |
| END | END |
| HOME | HOME |
| PAGE-DOWN | PAGE-DOWN |

**Table 5–3: Static Key Bindings in Windows** *(2 of 2)*

| Key Function | Key Label |
|---|---|
| PAGE-UP | PAGE-UP |
| RETURN | ENTER |
| TAB | TAB |

Table 5–4 lists the user-definable key bindings in the Windows environment.

**Table 5–4: User-definable Key Bindings in Windows** *(1 of 2)*

| Key Function | Key Label |
|---|---|
| APPEND-LINE | No default |
| BLOCK | No default |
| BREAK-LINE | No default |
| CLEAR | No default |
| DEFAULT-POP-UP | F4 |
| DELETE-LINE | No default |
| END-ERROR | ESC |
| FIND | No default |
| GET | No default |
| GO | F2 |
| HELP | F1 |
| INSERT-MODE | INS |
| NEW-LINE | No default |
| NEXT-FRAME | F6 |
| PREV-FRAME | SHIFT-F6 |

**Table 5–4: User-definable Key Bindings in Windows** *(2 of 2)*

| Key Function | Key Label |
|---|---|
| PUT | No default |
| RECALL | No default |

### Keymap Section

The Keymap section is an optional section you can create in the `progress.ini` file to define mappings between standard ASCII (7-bit) characters and extended (8-bit) characters. Extended characters are typically non-English alphabetical characters. Progress uses these keymap entries to build a translation table for processing characters in the input and output streams.

**NOTE:** Using the Keymap section is the Version 6 technique for mapping characters. Using conversion tables is a better technique for processing characters. For information about defining conversion tables, see the *Progress Internationalization Guide*.

Use the IN statement, in a keymap entry, to map an ASCII character to an extended character using the following syntax:

```
[ MAP-language ]
  :IN(\ascii-char)=\extended-char:
```

*language*

An identifier that appears with the MAP option.

*ascii-char*

The 7-bit character, in octal (*nnn*) format. You must specify all three digits.

*extended-char*

The 8-bit character, in octal (*nnn*) format. You must specify all three digits.

Use the OUT statement, in a keymap entry, to map an extended character to an ASCII character using the following syntax:

```
[ MAP-language ]
  :OUT(\extended-char)=\ascii-char:
```

You can use multiple IN and OUT statements in a single keymap entry.

If you use extended character sets, you must specify character mappings for each character set on a per-terminal basis. For example, the following code fragment demonstrates how to map characters to German language characters:

```
[MAP-german]
  :IN(\102)=\341:
  :IN(\101)=\216:
  :IN(\141)=\204:
  :IN(\117)=\231:
  :IN(\157)=\224:
  :IN(\125)=\232:
  :IN(\165)=\201:
```

The IN statements in the preceding example map ASCII characters (B, A, a, O, o, U, and u) to German characters (ß, Ä, ä, Ö, ö, Ü, and ü). When Progress sees the character A (\101) on input, it converts the character to Ä (\216). Likewise, when Progress needs to send an Ä to the terminal, it sends [ (\101). The terminal sees A and displays Ä.

Suppose the terminal cannot display an Ä. You can use the OUT statement to specify an appropriate character to display for Ä, such as capital A. For example:

```
[MAP-german]
  :OUT(\216)=\101:
```

You can also specify keymap entries for the input and output streams using the MAP option with each of the following Progress language elements:

- INPUT FROM statement
- INPUT THROUGH statement
- INPUT-OUTPUT THROUGH statement
- OUTPUT THROUGH statement
- OUTPUT TO statement

For more information about these language elements, see the *Progress Language Reference*.

### Specifying Environment Variables

Table 5–5 lists the environment variables you can set in either the Startup section or WinChar Startup section of the `progress.ini` file.

**Table 5–5: Environment Variables** *(1 of 2)*

| Variable | Description | Code Example |
|---|---|---|
| PROPATH | A list of directory paths separated by commas. When you run a procedure from within a Progress session with the RUN statement, Progress searches for it in the directory paths defined in PROPATH in the order listed.<br><br>**NOTE:** When Progress encounters a period in PROPATH (".,C . . ."), it substitutes the pathname of the current directory. | `PROPATH=.,C:\DLC` |
| PROMSGS | The full pathname of the Progress error messages file. The default value is `%DLC%\PROMSGS`. You only have to set the PROMSGS environment variable if you want to use an error messages file different from the default `PROMSGS` file in the `%DLC%` directory. | `PROMSGS=C:\DLC\PROLANG\GER\PROMSGS.GER` |
| PROCFG | The path name of your product's configuration file. The configuration file is a data file that identifies the Progress product and components that you are licensed to use. Reset PROCFG if you have moved the configuration file from the directory in which you installed Progress. | `PROCFG=C:\DLC\PROGRESS.CFG` |

**Table 5–5: Environment Variables** *(2 of 2)*

| Variable | Description | Code Example |
|---|---|---|
| PROBUILD | The PROBUILD End User Configuration utility uses this variable to qualify the pathnames in the link script PROBUILD produces. The default is `%DLC%\PROBUILD`. | `PROBUILD=C:\DLC\PROBUILD` |
| PROSTARTUP | The name of the Progress startup parameter file. The default is `startup.pf`. | `PROStartup= C:\DLC\PROSTARTUP.PF` |
| DLC | The pathname of the directory in which you installed the Progress system software. | `DLC=C:\DLC` |

## 5.1.4 Windows Icons for Running Progress

The install process places icons in the Progress group on the Windows Explorer desktop and in the Windows Start menu. The default group name is Progress; however, you can choose another group name. The icons vary depending on the products installed. Table 5–6 describes the icons.

**Table 5–6: Windows Installed Icons**

| Icon Title | Icon Command Line |
|---|---|
| Progress Server | *install-dir*\bin\mproserv.exe |
| Server Shutdown | *install-dir*\bin\proshut.exe |
| Progress Client | *install-dir*\bin\prowin32.exe |
| Progress Run-time Client | *install-dir*\bin\prowin32.exe -rr |
| Progress Help | winhelp.exe *install-dir*\prohelp\lgrfeng.hlp |
| Progress Desktop | *install-dir*\bin\prowin32.exe -p _desk.p |
| Progress AppBuilder | *install-dir*\bin\prowin32.exe -p _ab.p |

On the icon command line, *install-dir* is the directory where you have installed the products.

See the Windows user documentation for more information about creating additional icons, copying and modifying existing icons, and general information about the Windows Explorer.

### 5.1.5 Starting the Progress Client

You can start the Progress client using any of the following techniques:

- Choose the icon on the desktop.
- Choose Start→ Run from the task bar and enter startup parameters on the command line.
- Click on prowin32 from the Explorer.

Starting the Progress client places you in the Progress Procedure Editor with no databases attached.

### 5.1.6 Modifying Progress Client Icons

Follow these steps to modify startup parameters for a client icon:

1 ♦ Select the client icon from the Progress program group and choose File→ Properties from the Windows Explorer, or right-click on the icon and choose Properties from the pop-up menu. The Properties dialog box appears.

2 ♦ From the Shortcut tab, modify the command-line in the Target field. You can supply up to 259 characters in this field. The command-line limit of 259 characters is still valid for the expanded version of the command-line text.

3 ♦ Choose OK.

You can specify environment variables in the command-line. For example, specifying the following command runs the PROWIN32 executable located in the path where the %DLC% environment variable points:

```
%DLC%\BIN\PROWIN32.EXE
```

## 5.2 Maintaining the UNIX User Environment

Typically, you create a user-defined startup script to specify environment variable settings in the UNIX software environment. You can also use the `buildenv` script. Progress uses the `buildenv` script to maintain environment variables that define objects, libraries, and options required for building executables on UNIX.

You must also maintain the PROTERMCAP file. PROTERMCAP is a terminal definition file that contains the terminal description for your monitor's terminal type. Progress uses this file to determine how to interact with your terminal, console, or terminal emulator.

### 5.2.1 Maintaining the buildenv Script

The `buildenv` script contains settings for the following types of options:

- Options required by Progress
- Options required by Progress tools (such as the Procedure Editor and the PROBUILD utility)
- User-defined options
- Environment variables

Figure 5–3 shows a sample `buildenv` script.

```
#! /bin/sh

MACHINE="sun4"
PROLOAD=${PROLOAD-"$DLC/probuild"}
LDXOPT=${LDXOPT-"-x"}
LDOPT=${LDOPT-"-Bstatic"}
CCNOPT=${CCNOPT-""}
CC=${CC-"cc"}
SOCKLIB=${SOCKLIB-"-lsocket"}
TLILIB=${TLILIB-"-lns -lnsl"}
X11LIB=${X11LIB-"-lXt -lXext -lX11"}
CISAMLIB=${CISAMLIB-"lisam.a"}
SYBLIB=${SYBLIB-"libsybdb.a"}
ORALIB=${ORALIB-"$ORACLE_HOME/rdbms/lib/libocic.a
$ORACLE_HOME/rdbms/lib/osntab.o $ORACLE_HOME/rdbms/lib/libsqlnet.a
$ORACLE_HOME/rdbms/lib/libora.a"}
XPMLIB=${XPMLIB-"$DLC/probuild/4gl/libXpm.a"}
export MACHINE LDXOPT LDOPT CCNOPT CC
export CISAMLIB NETISAMLIB ORALIB X11LIB
export SOCKLIB TLILIB PROLOAD XPMLIB
```

**Figure 5–3: Sample buildenv Script on UNIX**

You can edit this script to customize environments at different sites. The `buildenv` script is located in the `$DLC/probuild/eucapp` directory. Progress uses the `buildenv` script to maintain environment variables that define objects, libraries, and options required for building executables on UNIX. For more information about building Progress executables, see Appendix A, "Building Progress Executables."

### 5.2.2 Maintaining the PROTERMCAP File

The terminal definition file for most character interfaces is called the PROTERMCAP file. This file is a flat file database that contains the terminal description for your monitor's terminal type to inform Progress on how to interact with your terminal, console, or terminal emulator. It specifies which characters to use to draw boxes, how to write in various video modes (such as inverse or blinking), how many lines are on the screen, and what actions to perform for each key on the keyboard.

**NOTE:** Make a backup copy of the PROTERMCAP file before making modifications.

Each terminal entry contains the commands and terminal descriptions that Progress uses to communicate with and control the terminal. Each entry is divided into these logical sections:

- Terminal name
- Terminal capabilities
- Progress terminal-specific key functions
- Progress color table
- Vermont Views key functions
- Pointer to key functions

The PROTERMCAP environment variable ($DLC/protermcap): points to the PROTERMCAP file.

The PROTERMCAP file contains a lengthy entry for each terminal type supported by Progress. Each entry has six parts. Figure 5–4 shows a partial example of each part of a PROTERMCAP entry.

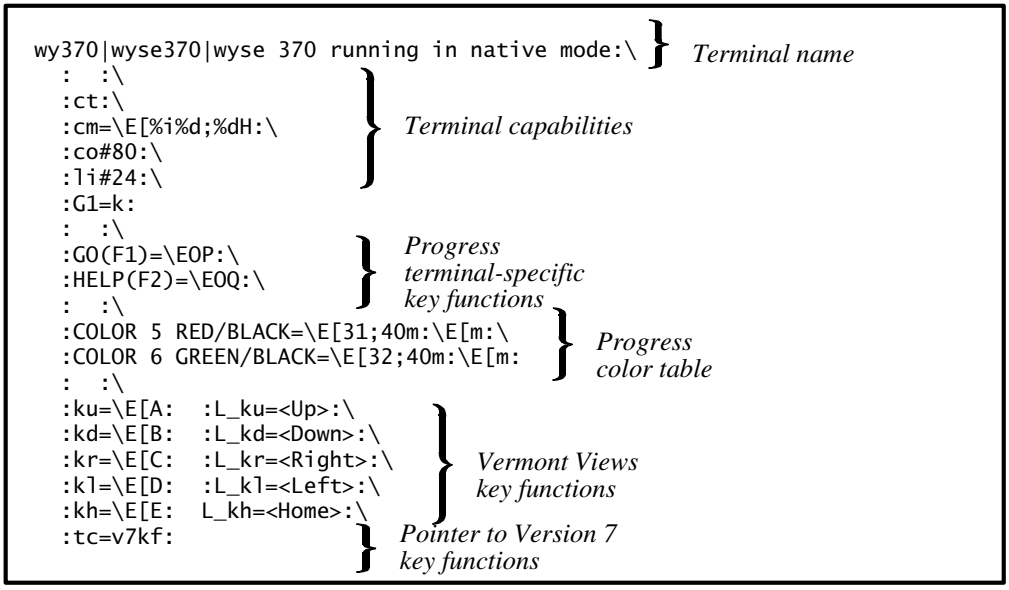



**Figure 5–4: Parts of a PROTERMCAP Entry**

**NOTE:** There is a 8,192-byte limit on the size of each PROTERMCAP entry.

The following sections describe PROTERMCAP syntax and each part of a PROTERMCAP entry.

### 5.2.3 PROTERMCAP Syntax

This section explains the following:

- The general syntax rules that apply when specifying fields in any of the six PROTERMCAP sections.
- The syntax rules that apply when specifying string values within a field. String values can occur in the terminal capabilities section, the key bindings section, and the color table section.

Syntax rules that apply only to specific sections of the PROTERMCAP file are explained in context.

### General Syntax

A PROTERMCAP entry is a single logical line that contains many fields. The syntax for an entry is as follows:

- Each field is terminated by a colon (:). (Use the octal equivalent \072 to specify a colon that is not a terminator.)
- Unless stated otherwise, a field cannot contain embedded separators. Spaces and tabs improve readability, but they are considered empty fields and must be terminated by a colon.
- Comment lines begin with a pound sign (#), and can only occur at the beginning and end of each PROTERMCAP terminal entry (that is, before the terminal name line and after the pointer to key functions line).
- To enhance readability, the PROTERMCAP file, as installed, contains as many physical lines as there are fields. Each physical line in the entry contains the following, in the order listed:
  - An empty field consisting of a tab character.
  - A field definition.
  - A backslash (\).
  - A return.

  The backslash indicates that the terminal entry continues on the next physical line of the file. If you need to include a literal backslash in a field, use two backslashes.

The following code fragment from the entry for the Wyse 370 shows the use of the colon, tab, space, backslash, and return characters:

```
wy370|wyse370|wyse_370|wyse 370 running in native mode:\
  :  :\
  :ct:\
  :cm=\E[%i%d;%dH:\
  :co#80:\
  :li#24:\
  :G1=k:
  :  :\
  :GO(F1)=\EOP:\
  :HELP(F2)=\EOQ:\
  :  :\
  :COLOR 5 RED/BLACK=\E[31;40m:\E[m:\
  :COLOR 6 GREEN/BLACK=\E[32;40m:\E[m:
```

### String Syntax

Certain fields in the capabilities, key bindings, and color sections are specified using strings. Strings are typically a combination of command sequences, literal values, and encoded arguments that tell Progress how to control some aspect of the display.

Table 5–7 show the symbols that are common to all strings. Symbols that are unique to strings specified in the capabilities, key bindings, or color sections are explained in context.

**Table 5–7: Syntax for Specifying String Values** *(1 of 2)*

| Mnemonic | Description |
|---|---|
| \E | ESC (ASCII 27) |
| \n | New line |
| \r | Return |
| \t | Tab |
| \b | Backspace |
| \f | Form feed |
| *999* | Octal representation of a character. A colon (:) within a string must be represented as \072 and a null character as \200. |

**Table 5–7: Syntax for Specifying String Values** *(2 of 2)*

| Mnemonic | Description |
|---|---|
| ^*x* | **CTRL-*x***, where *x* represents an appropriate character. For example, ^N represents **CTRL-N**. |
| \^ | ^ |
| \\ | \ |

Some terminal types require you to specify a delay period, also called a padding constant, with certain string capabilities. The delay, in milliseconds, gives the terminal adequate time to execute the command. Specify this delay as an integer value after the equals sign (=) in a string, as shown in the following example:

```
:cm=20\E[%i%d;%dH:\
```

This tells Progress to wait 20 milliseconds after sending the cursor motion (cm) command sequence.

You can optionally follow an integer delay value with an asterisk (*) for commands that affect more than one line of the display. In this case, Progress delays the specified time for each line affected by the command. However, this significantly degrades performance and usually is not required.

When you specify proportional delay, you must specify an integer as the delay in a string field. It cannot have a fractional part; for example, a delay of 1.5* is not valid.

**NOTE:** Do not specify a padding constant with the GS, GH, GV, GE, G1, G2, G3, and G4 capabilities.

### 5.2.4 Terminal Name Entries

The first field in each PROTERMCAP terminal entry is the Name field. This field provides the names known for the terminal. The Name field contains one or more of the following components, separated by the pipe symbol ( | ):

- An abbreviation of the terminal's name.
- The common name of the terminal.
- Additional names for the terminal, if any.

- A description of the terminal in which only the last name can contain blanks for readability (optional).

PROTERMCAP requires at least one of the name entries (1, 2, or 3, above) to be entered without spaces; the other two, and the description, are optional. See the following examples.

For example:

```
wy370|wyse370|wyse_370|wyse 370 running in native mode:\
```

*Abbreviation* *Common Name* *Additional Name* *Description*

Another example of a terminal name entry is:

```
wy370|wyse 370 running in native mode:\
```

*Abbreviation* *Description*

### 5.2.5 Terminal Capabilities Entries

The terminal capabilities section of a terminal entry contains fields that tell Progress features of the terminal being used. These entries are indented from the name line to distinguish them from the terminal name.

Terminal capabilities entries control visual attributes of the terminal: how big the terminal screen is, how to move to any point on the screen, how to refresh the screen, how to enter/exit special display modes (reverse, blinking, underline), etc. They also control specific terminal functions to set up and clean up the terminal properly, so as not to leave a terminal in an unexpected state (which might leave the terminal unusable).

These entries are the core sequences that Progress uses to communicate with the terminal. Changing them can cause very unpredictable results and might even cause Progress to put the terminal in an unusable state. The entries can be changed, but exercise great care in doing so.

Terminal capability entries consist of:

- A two-character mnemonic that identifies a generic terminal capability
- An assignment operator equal sign (=), or a pound sign (#)

- For certain capability types, a value that allows Progress to understand and interact with the capability on a specific terminal

For example:

```
:ct:\
:cm=\E[%i%d;%dH:\
:co#80:\
:li#24:\
:G1=k:\
```

### Functional Listing of Terminal Capabilities

These are the cursor terminal capabilities that Progress supports:

bc, CF, CN, cm, kd, kl, kr, ku, ke, ks

These are the color/video terminal capabilities that Progress supports:

BB, BR, ct, HS, HR, se, so, ue, us

These are the graphics character terminal capabilities that Progress supports:

G1, G2, G3, G4, GH, GV, GE, GS

**NOTE:** Do not use a padding constant with any of the above graphic strings.

These are miscellaneous terminal capabilities that Progress supports:

co, li, CA, ce, cl, is, Se, Si, te, ti, pc, xi

These are the key translation capabilities that Progress supports:

k0, k1, k2, k3, k4, k5, k6, k7, k8, k9, k0, k. (period), k, (comma), k- (hyphen)

These are the scroll region capabilities that Progress supports:

al, sf, sr, dl, cs

**NOTE:** The ce, cl, and cm capabilities are required for Progress to start up.

Vermont Views supports all the capabilities listed here, except ce, te, and ti, and the key translation capabilities.

Table 5–8 describes the mnemonics that Progress supports.

**Table 5–8: Alphabetical Listing of Capability Mnemonics** *(1 of 4)*

| Mnemonic | Data Type | Description |
|---|---|---|
| al | STRING | Add line. |
| bc | STRING | Backspace character. |
| BB | STRING | Enter COLOR 4 mode (usually blink mode). |
| BR | STRING | Exit COLOR 4 mode (usually blink mode). |
| CA | BOOLEAN | Terminal clears screen with current attribute. If you do not specify and this capability is available, full-screen clears with attributes other than NORMAL will be slow. |
| ce | STRING | Clear to end of line. |
| CF | STRING | Cursor off. |
| cl | STRING | Clear screen. This improves speed. On some terminals, you must define cl as home cursor followed by clear to end of screen. |
| cm | STRING | Cursor motion. |
| CN | STRING | Cursor on. |
| co | INTEGER | Columns on screen (usually 80). |
| cs | STRING | Change scroll region. Uses cm syntax. |
| ct | BOOLEAN | Terminal supports color. |
| dl | STRING | Delete line. |
| G1 | STRING | Single-line upper-right corner character. |
| G2 | STRING | Single-line upper-left corner character. |
| G3 | STRING | Single-line lower-left corner character. |
| G4 | STRING | Single-line lower-right corner character. |

**Table 5–8: Alphabetical Listing of Capability Mnemonics** *(2 of 4)*

| Mnemonic | Data Type | Description |
|---|---|---|
| GE | STRING | Graphics end. Sent when finished drawing box or underline. |
| GH | STRING | Single-line horizontal character. |
| GS | STRING | Graphics start. Sent to begin drawing box or underline. |
| GV | STRING | Vertical-line graphic character. |
| HR | STRING | Exit COLOR 3 mode (usually highlight mode). |
| HS | STRING | Enter COLOR 3 mode (usually highlight mode). This is usually set to high intensity, if available. |
| is | STRING | Terminal initialization string. Sent when Progress starts. |
| k0-9<br>k.<br>k,<br>k- | STRING | Define the codes sent by the numeric keypad keys if these are different from the codes sent by the standard 0-9, period, comma, and hyphen keys. These are only used by Progress. |
| kd | STRING | Down arrow. |
| ke | STRING | Exit keypad mode. |
| kl | STRING | Left arrow. |
| kr | STRING | Right arrow. |
| ks | STRING | Set keypad mode. |
| ku | STRING | Up arrow. |
| li | INTEGER | Lines on screen (usually 24). |
| pc | CHARACTER | Pad character (defaults to null). |
| Se | STRING | String to send when Progress terminates (after te). |
| se | STRING | Exits COLOR 2 mode, or MESSAGE color (usually reverse video). |

**Table 5–8: Alphabetical Listing of Capability Mnemonics** *(3 of 4)*

| Mnemonic | Data Type | Description |
|---|---|---|
| sf | STRING | Scroll forward. You can use this on terminals that have scrolling regions (for example, terminals with cs and sr defined). |
| Si | STRING | String to send when Progress starts (after is). |
| so | STRING | Enter COLOR 2 mode, or MESSAGE color (usually reverse video). Progress uses this attribute by default for the two-line message area at the bottom of the screen. It is also the default for the PROMPT-FOR color of selectable widgets (that is, buttons, sliders, toggle-boxes, etc.). If not set and the PROMPT-FOR color of a widget is not explicitly set in the 4GL, a widget might not be visible. |
| sr | STRING | Scroll reverse. |
| te | STRING | Cursor movement string to send when Progress terminates. |
| ti | STRING | Cursor movement string to send when Progress starts. |
| ue | STRING | Exit COLOR 1 mode, or INPUT color. |
| us | STRING | Enter COLOR 1 mode, or INPUT color (usually the underline attribute). Progress uses COLOR 1 as the default PROMPT-FOR color for fill-ins and editors. If COLOR 1 is not defined by the us and ue capabilities, and the PROMPT-FOR color for fill-ins and editor widgets is not explicitly set in the 4GL, these widgets might not be visible, and there might be no indication they are enabled. |

**Table 5–8: Alphabetical Listing of Capability Mnemonics** *(4 of 4)*

| Mnemonic | Data Type | Description |
|---|---|---|
| ws | BOOLEAN | Uses the device /dev /tty for the current port to determine the number of columns and rows for the terminal. Meaningful primarily for emulated terminals in a windowing system (that is, xterm). If successful, overrides co and li; otherwise, defaults to co and li. Progress will not automatically resize if the size of the terminal changes after initialization. |
| xi | BOOLEAN | Terminal will not automatically do a hardware scroll when the last position (last row and last column) on the screen is written to. Specify whether available, otherwise Progress will not write to the last position in order to avoid the automatic scrolling. |

There are four types of terminal capability fields: BOOLEAN, NUMERIC, CHARACTER, and STRING. Table 5–9 summarizes the four data types and the operators used in conjunction with each. Boolean fields are specified by a mnemonic abbreviation only. The mnemonic indicates that the feature is present and its absence indicates that the feature is not present. All other capabilities take a value. The value can be a string, a character, or a numeric value.

**Table 5–9: Data Types and Operators**

| Data Type | Operator |
|---|---|
| BOOLEAN | none |
| NUMERIC | # and value |
| CHARACTER | = and character |
| STRING | = and string |

The following code fragment from the terminal entry for the Wyse 370 terminal shows how to specify BOOLEAN, NUMERIC, CHARACTER, and STRING capabilities:

```
:ct:\
:cm=\E[%i%d;%dH:\
:co#80:\
:li#24:\
:G1=k:\
```

In this example, the mnemonic ct is a Boolean switch that indicates that the terminal supports color. The mnemonic cm specifies cursor motion and takes a string. The assignment operator for string values is an equals sign (=). The mnemonics co and li specify the number of columns and lines on the display and each takes an integer. The assignment operator for integer values is a pound sign (#). The mnemonic G1 specifies the upper-right corner line-graphic character and takes a character. The assignment operator for character values is an equals sign (=).

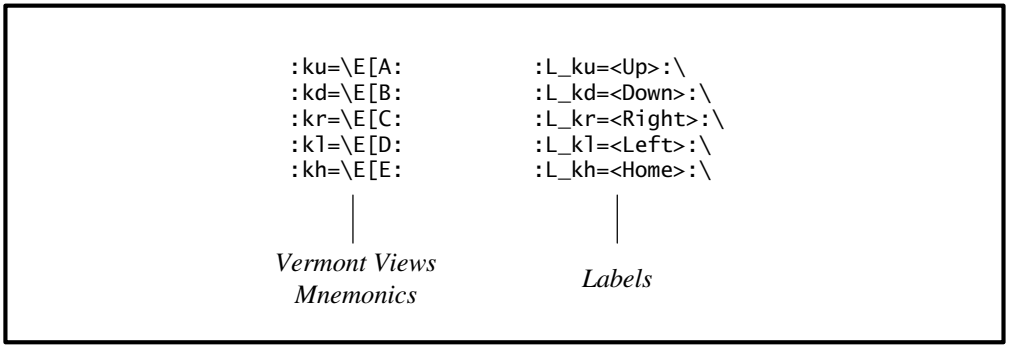
### 5.2.6 Vermont Views Key Function Capabilities

Certain programs, such as the install program, use a character interface library called Vermont Views for character input. Vermont Views does not use the Progress Key Function syntax of keyfunction(label)=sequence. Instead, it uses key function and key label capabilities. Except for bc, ku, kd, kl, and kr, these key function capabilities are unique to Vermont Views. The key label capabilities mnemonics are created from the key function capabilities mnemonics by prepending “L_”. Because both Progress and Vermont Views use only the first instance of a capability, each key function capability can represent only one key label.

For readability, the Vermont Views section of an entry has two distinct columns:

- The first lists the key function capabilities.
- A list of the matching key label capabilities. The label in this column always starts with “L_”.

Here is an example:




**NOTE:** For readability in the PROTERMCAP file, the bc, ku, kd, kl, and kr capabilities are usually repeated in the Vermont Views part of the file, even though both Progress and Vermont Views use these capabilities as they appear earlier in the Progress capabilities part. To avoid confusion, the key sequences for the Vermont Views capabilities should be the same as the Progress capabilities.

Table 5–10 summarizes the Vermont Views key function capabilities mnemonics.

**Table 5–10: Vermont Views Key Function Mnemonics** *(1 of 2)*

| Mnemonic | Description |
|---|---|
| kh | Home (beginning of line, then beginning of page) |
| EN | End (end of line, then end of page) |
| PU | Page up |
| PD | Page down |
| ki | Insert toggle |
| DL | Delete character |
| ESC | Exit current form |
| bt | Back tab |
| fk0 | Function Key 1 |
| fk1 | Function Key 2 (contents) |

**Table 5–10: Vermont Views Key Function Mnemonics** *(2 of 2)*

| Mnemonic | Description |
|---|---|
| fk2 | Function Key 3 (enter the menu bar, then exit the menu bar) |
| fk3 | Function Key 4 (exit current form) |
| fk4 | Function Key 5 (browse backward) |
| fk5 | Function Key 6 (browse forward) |
| fk6 | Function Key 7 (go back through hypertext links) |
| fk7 | Function Key 8 (bring up search dialog box) |
| fk8 | Function Key 9 |
| fk9 | Function Key 10 |
| Aka | Browse backward |
| Akd | Browse forward |
| Aki | Exit current form |
| Akp | Go back through hypertext links |
| Aks | Bring up search dialog box |
| Aku | Menu bar |
| Akw | Contents |

### 5.2.7 Pointer to Key Functions

The last line of every terminal entry is a pointer to the Progress key functions. These key functions are common to all terminal types and are used by Progress to map certain key functions with certain control sequences, for example, **:FIND (CTRL-F) = ^F:**. This pointer always appears as follows:

```
tc=v7kf:
```

If you modify or create new PROTERMCAP entries, be sure to include this line as the last line in the entry. This line ensures that the terminal entry can support Progress Version 7 and later.

If you want to use Version 6 mappings with Progress Version 7 and later, change the pointer as follows:

```
tc=v6kf:
```

### 5.2.8 Specifying Colors

You specify colors in the Progress terminal-specific key functions and capabilities section of PROTERMCAP. You can specify up to 123 color fields in a terminal entry. This manual refers to the color specifications as the color table.

Each line in the table performs two semantic functions: it defines a foreground/background color pair and it maps that color pair to an integer from 5 to 127. The integer is how an application references the color pair when making color assignments to a widget. (An application typically assigns two color pairs to a widget: one for display mode and another for prompt mode.)

Use the following syntax to specify a color field in a PROTERMCAP terminal entry:

```
COLOR color-number [color-name]=start-sequence:stop-sequence:
```

*color-number*

An integer from 5 to 127 that specifies the location of the color in the color table. The color number is the mechanism used to assign a color pair in an application.

*color-name*

The name of the color. You can use any color value except a Progress keyword. Although this is optional, including a name makes the color section of the terminal entry self-documenting (you cannot embed comments) and also makes the color specifications backwardly compatible.

*start-sequence*

The character sequence that starts the color attribute.

*stop-sequence*

The character sequence that stops the color attribute.

For example, the following fields from the Wyse 370 terminal define color table locations 5 and 6:

```
COLOR 5 RED/BLACK=\E[31;40m:\E[m:\
COLOR 6 GREEN/BLACK=\E[32;40m:\E[m:\
```

Progress reserves color table locations 0 to 4 as follows:

- Color 0 holds the Version 6 color NORMAL, which is set by the terminal initialization (is) field.
- Color 1 holds the Version 6 color INPUT and is set and cleared by us and ue. As installed, this is underline mode.
- Color 2 holds the Version 6 color MESSAGE and is set and cleared by so and se. As installed, this is reverse-video.
- Color 3 holds the colors used for high intensity and is set and cleared by HS and HR.
- Color 4 holds the colors used for blink and is set and cleared by BB and BR.

**NOTE:** The PROTERMCAP file, as installed, does not support spacetaking (embedded attribute) terminals. These are terminals that use characters on the display to hold control codes for video and color capabilities. If you add support for such terminals, Progress ignores the capability when displaying the interface.

### 5.2.9 Specifying the Cursor Motion Capability

The absolute cursor-motion (cm) capability allows Progress to move to any row and column on the display. The capability takes a string that is composed of two parts:

- Control characters
- Two conversion specifications

The conversion specifications allow Progress to convert an integer row value and an integer column value into the arguments the terminal expects.

The cm string syntax is exactly the same as that used in the standard UNIX `termcap` file, in which the conversion specifications follow the pattern of the UNIX `printf()` command. Table 5–11 lists the supported conversion specifications, each of which begins with a percent sign (%).

**Table 5–11: Syntax for Specifying Cursor Motion String Value**

| Mnemonic | Description |
|---|---|
| %d | Send the integer to the terminal in ASCII format. |
| %2 | Pad the value with a leading blank if it is only one digit. |
| %3 | Pad the value with up to two leading blanks. |
| %. | Treat the value as an ASCII character value. |
| %+x | Value += x, then does a %. |
| %>xy | If value > x, then value +=y; no output. |
| %r | Reverses the order of row and column; no output. |
| %i | Allows row and column values based on a zero origin to be incremented by one to make them appropriate for terminals whose origin is at (1,1). |
| %% | Gives a single %. |
| %n | Exclusive OR row and column with 0140. |
| %B | BCD (16*(value/10)) + (value%10); no output. |
| %D | Reverse coding (value - 2*(value%16)); no output. |
| %x | Gives a two-byte, zero-padded, hex value as in the "%0.2x" format for the UNIX `printf()` function. |
| %M | This conversion specification is not similar to any UNIX printf() format.<br>%Mx%Mx translates to:<br>p1 = row / x + x<br>p2 = row mod x + x<br>p3 = col / x + x<br>p4 = col mod x + 2x<br>where p1 through p4 are sent to the terminal as characters. |

The following example is the cursor motion field for the wyse 370 terminal:

```
:cm=\E[%i%d;%dH:\
```

- **\E[** — The standard ANSI console lead-in sequence (Escape, followed by a left square bracket). Progress transmits this literally.
- **%i** — Instructs Progress to add one to the row and column values to compensate for a terminal whose origin is 1,1.
- **%d** — Instructs Progress to send the row and column values to the terminal in ASCII format.
- **;** — Separates the row value from the column value.
- **H** — A terminating character.

To create a cm string for a new terminal type, find the description of this capability in your terminal's documentation to determine the control characters and the arguments it expects to move the cursor to a random spot on the display. The documentation specifies an algorithm for converting a row/column integer pair to the arguments it needs. The manual also specifies the order in which the row and column values are expected (typically row first), and the character to use to separate the two values. For example, to move to column 65, row 66, a terminal might expect the characters A and B, which have ASCII values of 65 and 66, respectively.

### 5.2.10 Specifying Keyboard Mappings

The PROTERMCAP file maps key functions to key labels. Table 5–12 shows the mappings as installed. Many of the mappings shown are terminal specific, and certain terminals might not support the function for a particular sequence given in the table.

**Table 5–12: Progress PROTERMCAP Key Functions** *(1 of 7)*

| Key Function | UNIX Key Label |
|---|---|
| ABORT | CTRL-\<br>CTRL-ALT-DEL |
| APPEND-LINE | CTRL-A |

**Table 5–12: Progress PROTERMCAP Key Functions** *(2 of 7)*

| Key Function | UNIX Key Label |
|---|---|
| BACKSPACE | BACKSPACE<br>CTRL-H<br>DEL-CHAR |
| BACK-TAB | BACK-TAB<br>CTRL-U<br>SHIFT-TAB |
| BELL | BELL<br>CTRL-G |
| BLOCK | SELECT<br>CTRL-V |
| BOTTOM-COLUMN | ESC-CTRL-B |
| BREAK-LINE | ESC-B |
| CANCEL-PICK | ESC-CTRL-X |
| CHOICES | ESC-CTRL-H |
| CLEAR | F8<br>CTRL-Z |
| CLOSE | ESC-Z |
| COMPILE | ESC-P |
| COPY | F11<br>ESC-C |
| CURSOR-DOWN | CURSOR-DOWN<br>CTRL-J |
| CURSOR-LEFT | CURSOR-LEFT<br>CTRL-O |
| CURSOR-RIGHT | CURSOR-RIGHT<br>CTRL-L |

**Table 5–12: Progress PROTERMCAP Key Functions** *(3 of 7)*

| Key Function | UNIX Key Label |
|---|---|
| CURSOR-UP | CURSOR-UP<br>CTRL-K |
| CUT | F10<br>ESC-X |
| DEFAULT-POP-UP | ESC-U |
| DELETE-CHARACTER | DEL |
| DELETE-COLUMN | ESC-CTRL-Z |
| DELETE-END-LINE | ESC-K |
| DELETE-FIELD | ESC-CTRL-D |
| DELETE-LINE | REMOVE<br>CTRL-D |
| DELETE-WORD | ESC-D |
| EDITOR-BACKTAB | CTRL-B |
| EDITOR-TAB | CTRL-G |
| END | END<br>ESC ->
ESC-. |
| END-ERROR | F4<br>PF4<br>ESC<br>CTRL-E |
| ENTER-MENUBAR | F3<br>PF3<br>ESC-M |
| EXIT | ESC-Q |
| FIND | CTRL-F<br>FIND |

**Table 5–12: Progress PROTERMCAP Key Functions** *(4 of 7)*

| Key Function | UNIX Key Label |
|---|---|
| FIND-NEXT | ESC-F |
| FIND-PREVIOUS | ESC-I |
| GET | F5<br>ESC-O |
| GO | F1<br>PF1<br>CTRL-X<br>DO |
| GOTO | ESC-G |
| HELP | HELP<br>F2<br>ESC-?<br>PF2 |
| HOME | HOME<br>ESC-H<br>ESC-h<br>ESC-<<br>ESC-, |
| INSERT-COLUMN | ESC-CTRL-N |
| INSERT-FIELD | ESC-CTRL-G |
| INSERT-FIELD-DATA | ESC-CTRL-F |
| INSERT-FIELD-LABEL | ESC-CTRL-E |
| INSERT-MODE | INS<br>F9<br>CTRL-T<br>CTRL-@ |
| LEFT-END | ESC-LEFT-ARROW |
| MAIN-MENU | ESC-CTRL-M |

**Table 5–12: Progress PROTERMCAP Key Functions** *(5 of 7)*

| Key Function | UNIX Key Label |
|---|---|
| MOVE | ESC-CTRL-V |
| NEW | ESC-N |
| NEW-LINE | INSERT-HERE<br>CTRL-N |
| NEXT-ERROR | ESC-E |
| NEXT-FRAME | ESC-CTRL-I |
| NEXT-WORD | CTRL-W |
| OPEN-LINE-ABOVE | ESC-L |
| OPTIONS | ESC-CTRL-O |
| PAGE-DOWN | PAGE-DOWN<br>ESC-CURSOR-DOWN<br>ESC-DOWN-ARROW<br>NEXT-PAGE<br>NEXT-SCRN |
| PAGE-LEFT | ESC-W |
| PAGE-RIGHT | ESC-Y<br>ESC-CTRL-J |
| PAGE-RIGHT-TEXT | ESC-CTRL-J |
| PAGE-UP | PAGE-UP<br>ESC-CURSOR-UP<br>ESC-UP-ARROW<br>PREV-PAGE<br>PREV-SCRN |
| PASTE | F12<br>ESC-V |
| PICK | ESC-CTRL-P |
| PICK-AREA | ESC-CTRL-W |

**Table 5–12: Progress PROTERMCAP Key Functions** *(6 of 7)*

| Key Function | UNIX Key Label |
|---|---|
| PICK-BOTH | ESC-CTRL-Q |
| PREV-FRAME | ESC-CTRL-U |
| PREV-WORD | CTRL-P |
| PUT | F6<br>ESC-S |
| RECALL | F7<br>CTRL-R |
| REPLACE | ESC-R |
| REPORTS | ESC-CTRL-A |
| RESUME-DISPLAY | CTRL-Q |
| RETURN | RETURN<br>CTRL-M |
| RIGHT-END | ESC-CURSOR-RIGHT<br>ESC-RIGHT-ARROW |
| SAVE-AS | ESC-A |
| SCROLL-LEFT | ESC-CTRL-L |
| SCROLL-MODE | ESC-T |
| SCROLL-RIGHT | ESC-CTRL-R |
| SETTINGS | ESC-CTRL-@ |
| STOP | CTRL-C<br>CTRL-BREAK |
| STOP-DISPLAY | CTRL-S |

**Table 5–12: Progress PROTERMCAP Key Functions** *(7 of 7)*

| Key Function | UNIX Key Label |
|---|---|
| TAB | TAB<br>CTRL-I |
| TOP-COLUMN | ESC-CTRL-T |

### Using Key Function Syntax

To change any of these mappings or create new ones, use the following syntax:

```
keyfunction(key-label) = sequence:
```

*keyfunction*

The name of a key function.

*key-label*

The key label as it appears on the keyboard.

*sequence*

The characters transmitted when the key is pressed.

As in other sections of the PROTERMCAP file, string values are assigned using an equal sign (=) and the field is terminated with a colon (:).

For example:

```
RECALL(F7)=^AF\r:
```

This field in a PROTERMCAP terminal entry defines the **F7** key as transmitting a **CTRL-A** followed by a capital **F** followed by a carriage return, and associates use of **F7** with the RECALL function.

The following field defines **SCROLL-DOWN** to act like **PAGE-DOWN**:

```
PAGE-DOWN(SCROLL-DOWN)=\021:
```

If you assign the same key label to two or more different key functions, you get a warning message when you start Progress. For example, "You cannot use DELETE for both DELETE-CHARACTER and BACKSPACE."

If you use a key in the Progress ON statement or GO-ON option only, and do not have to assign it to a standard action, then use the following syntax in the PROTERMCAP entry:

```
(key-label)=sequence:
```

For example, the following entry indicates that pressing **F16** sends a capital O followed by a carriage return:

```
(F16)=O\r:
```

If any of the control code sequences sent when you press a key on the keyboard begin with a control key, you cannot use that control key on your keyboard and the key does not have its normal Progress meaning. For example, if you specify **CTRL-F** in a control code sequence when creating a key mapping, you can no longer use **CTRL-F** for **FIND**. You have to map another key to the **FIND** action.

The key labels that the UNIX `stty` command specifies for **FLUSH** and **SUSPEND** override the PROTERMCAP file's use of the same key labels. For example, if the `stty` settings for **FLUSH** and **SUSPEND** are **CTRL-Q** and **CTRL-S**, you cannot map these key labels to key functions in the PROTERMCAP file. If you do, you receive no warning; the labels assume their `stty` meanings at run time and Progress ignores them.

### UNIX stty Control Functions

The installed PROTERMCAP file does not map the Progress functions ABORT, STOP, and UNIX-END to key labels. Progress instead uses the key labels that the UNIX `stty` command specifies for **QUIT**, **INTR** and **EOF** respectively.

For example, the following `stty` command specifies that Progress should use **CTRL-** for ABORT, **CTRL-C** for STOP, and **CTRL-D** for UNIX-END:

```
stty quit ^\ intr ^C eof ^D
```

The labels specified by the `stty` command are of two forms: either **CTRL-X** or DEL if the **DELETE** key (octal 177, decimal 127) is used. When entering the `stty` command, indicate the control character by holding down the **CTRL** key and pressing the specified key; you do not type a caret (^) followed by the key.

If an entry in the PROTERMCAP file uses one of the key labels specified in the `stty` command, you get a warning message when you start Progress. For example, if the `stty` command specifies the **DELETE** key for the STOP function and the PROTERMCAP file specifies the **DELETE** key for the DELETE-CHARACTER function, you receive a warning message.

In UNIX environments that do not use the Bourne shell (for example, the Korn shell or C shell), job control allows you to end a job currently executing on a terminal. In most environments this is initiated using **CTRL-Z**; however, Progress uses this character sequence to clear the editor.

### 5.2.11 Extended Character Support Entries

You can define mappings between standard ASCII (7-bit) characters and extended (8-bit) characters in the PROTERMCAP file. Extended characters are typically non-English alphabetical characters. Progress uses these mappings to build a translation table for processing characters in the input and output streams.

If you use extended character sets, you must specify character mappings for each character set on a per-terminal basis. For example, the following code fragment from the PROTERMCAP file demonstrates how to map characters on the Wyse 60 keyboard to German language characters:

```
ger|german|wy60 in german mode:\
  :IN(\102)=\341:\
  :IN(\101)=\216:\
  :IN(\141)=\204:\
  :IN(\117)=\231:\
  :IN(\157)=\224:\
  :IN(\125)=\232:\
  :IN(\165)=\201:
```

The IN statements in the preceding example map ASCII characters (B, A, a, O, o, U, and u) to German characters (ß, Ä, ä, Ö, ö, Ü, and ü). When Progress sees the character A (\101) on input, it converts the character to Ä (\216). Likewise, when Progress needs to send an Ä to the terminal, it sends [ (\101). The terminal sees A and displays Ä.

Suppose the terminal cannot display an Ä. You can use the OUT statement to specify an appropriate character to display for Ä, such as capital A. For example:

```
:OUT(\216)=\101:
```

The IN and OUT statements express both the ASCII character and the extended character in octal (*nnn*) format.

Given the appropriate PROTERMCAP file entries for a language and terminal, set the TERM environment variable to *terminal/language* (for example, TERM=wy60/german).

You can also specify character mappings for the input and output streams using the MAP option with each of the following Progress language elements:

- INPUT FROM statement
- INPUT THROUGH statement
- INPUT-OUTPUT THROUGH statement
- OUTPUT THROUGH statement
- OUTPUT TO statement

For more information about these language elements, see the *Progress Language Reference*.

### 5.2.12 Creating a PROTERMCAP Entry

On UNIX, if Progress does not support your terminal or you need to modify the PROTERMCAP entry for your terminal, refer to the programming documentation from your terminal's manufacturer. It should specify the appropriate commands, values, and key sequences that Progress needs to interact with your terminal.

If you need to create a new PROTERMCAP entry, first check the `/etc/termcap` file for an entry for your terminal. The `/etc/termcap` file is the standard terminal definition file used by several UNIX system programs, such as the vi editor. This file is usually provided with UNIX.

If this file contains an entry for your terminal, copy this entry to the PROTERMCAP file. Use this file as a starting point to build an entry for your terminal in the PROTERMCAP file.

If your version of UNIX uses a personal `termcap` called `terminfo`, you must modify the `terminfo` keywords to match the PROTERMCAP capability mnemonics described in the preceding sections.

### 5.2.13 Testing Terminal Entries

If you left any required specifications out of the PROTERMCAP file, when you run Progress you will receive an error message. Once in the Progress Procedure Editor, try some operations, such as insert line, delete line, insert character, delete character, and scroll up/down. Also run procedures that require data entry and display messages in the message area. To fully test the terminal:

- Go into the Progress Procedure Editor.

- Test to see that the key-functions and key-labels work.
- Test to see that colors are working correctly on widgets and frames.
- Test the output to terminal paged to test scrolling capabilities.

### 5.2.14 Setting Up the Terminal

Use your terminal's setup mode to synchronize the terminal with the color and key code definitions in its PROTERMCAP entry.

Progress supports terminals based on definitions of their characteristics stored in a file called the PROTERMCAP file. The PROTERMCAP file supplied with Progress has built-in definitions for many terminals. If your terminal is not included in the PROTERMCAP file supplied with Progress, you must make modifications to that file. Progress uses the shell variable or logical PROTERMCAP to find the PROTERMCAP file. By default, Progress looks for this file whenever you start the software in a character environment.

### 5.2.15 Setting the Terminal Type

Use the appropriate command in Table 5–13 to make a new terminal type the current terminal type. The commands assume that the name of the new terminal type is my_term_type.

**Table 5–13: Setting the Terminal Type**

| Operating System | Command | Environment |
|---|---|---|
| UNIX (Bourne shell) | `TERM="my_term_type"; export TERM` | In `.profile` or at the shell prompt |
| UNIX (C shell) | `setenv TERM "my_term_type"` | In `.profile` or at the shell prompt |

Alternately, you can change the terminal type from within a Progress program with the following Progress statement:

```
TERMINAL = termid
```

See the TERMINAL Function reference entry in the *Progress Language Reference* for more information.

**NOTE:** Terminal names are case sensitive.

### 5.2.16 Setting the PROTERMCAP Environment Variable

If you create a new PROTERMCAP file, store it in the current working directory. Then set the PROTERMCAP environment variable, using the appropriate command from Table 5–14, to allow Progress to use the new file.

**Table 5–14: Setting the PROTERMCAP Environment Variable**

| Operating System | Command |
|---|---|
| UNIX (Bourne shell) | `PROTERMCAP=protermcap; export PROTERMCAP` |
| UNIX (C shell) | `setenv PROTERMCAP protermcap` |

### 5.2.17 Reverting to the Default PROTERMCAP File

Use the appropriate command in Table 5–15 at the operating system prompt to go back to using the default PROTERMCAP file.

**Table 5–15: Reverting to the Default PROTERMCAP File**

| Operating System | Command |
|---|---|
| UNIX (Bourne shell) | `unset PROTERMCAP` |
| UNIX (C shell) | `unsetenv PROTERMCAP` |

### 5.2.18 Copying an Existing Terminal Entry

Use the `tc` (terminal copy) command when you are defining a new terminal entry that is similar to an existing entry. The command appends the specifications for the similar terminal to the new entry, so you do not have to retype the similar information.

The `tc` command takes a string, as shown in the following example:

```
:tc=terminal-name
```

Follow these guidelines when using the `tc` command:

- To override a capability in the copied entry, specify the capability in the new terminal's entry. Commands specified first always override those specified later.

- To suppress the inclusion of a capability that you do not want to specify in the new terminal's entry, specify only the mnemonic, followed by an at symbol (@) in the new terminal's entry. This tells Progress to ignore the capability when it reads it from the similar terminal.
- You cannot use the Version 6 PROTERMCAP in Version 7 or later. You can edit the PROTERMCAP to use the Version 6 key bindings by editing the following line at the end of the file:

```
:tc=v7kf {change to} :tc=v6kf
```

- Make `tc` the last field in the terminal entry.
- Place the similar terminal entry in the PROTERMCAP file before entries that reference it.
- The combined length of all the definitions must be less than 8,192 bytes.

For example, the following code fragment is a terminal entry for a Wyse 370 in 132 column mode. The entry specifies the name of the terminal, a terminal initialization sequence, the number of columns, and then uses `tc` to append the specifications for the Wyse 370 terminal:

```
wy370-132:Wyse 370 in 132 column mode:\
  :is=\E[?3h\E[90;1"p\E(B\E)0\E[63;0w:\
  :co#132:
  :tc=wy370:
```

# 6

# Managing R-code Libraries

A Progress r-code library is a collection of r-code procedures combined in a single file. It is similar to an object library in DOS or an archive file in UNIX. However, object libraries and archive files are used solely for efficient storage. R-code libraries allow you to manage and execute r-code procedures more efficiently.

This chapter provides information about the following topics:


- "Library Overview"
- "Setting Up a Library"
- "Running Procedures from a Library"
- "Libraries and PROPATH"
- "Memory and Network Considerations"
- "Using the PROLIB Utility"
- "Code Page Compatibility"

## 6.1 Library Overview

Progress provides two types of r-code libraries: standard and memory-mapped. A standard library contains r-code procedures that execute in local memory. A memory-mapped library contains r-code procedures that execute in shared memory.

When you execute r-code procedures from either a standard library in local memory or a memory-mapped library in shared memory, you gain the following advantages:

- Faster access to r-code
- Fewer file open and close operations
- Less I/O to temporary files

When you execute r-code procedures from a memory-mapped library in shared memory, you gain the following additional advantages:

- Even faster access to r-code
- Requires less memory because multiple users access the same r-code segments in shared memory

When loading and executing r-code procedures from operating system files in a directory, Progress must open and close each file individually. When loading and executing r-code procedures from standard or memory-mapped libraries, Progress opens only one file—the library itself—to access all of the r-code files in the library.

The r-code files that you place in a library are called *members*. Progress opens a library the first time you run a member procedure from the library. The library stays open until the end of your Progress session or until you remove the library from the PROPATH. For more information about how standard and memory-mapped libraries interact with PROPATH during a Progress session, see the "Libraries and PROPATH" section later in this chapter.

For information about monitoring and optimizing r-code execution during a Progress client session, see Chapter 4, "Managing Client Performance." For more information about r-code structure and execution, see the *Progress Programming Handbook*.

### 6.1.1 Loading R-code from a File

Figure 6–1 shows the load operation when executing a standard r-code procedure from an operating system file.

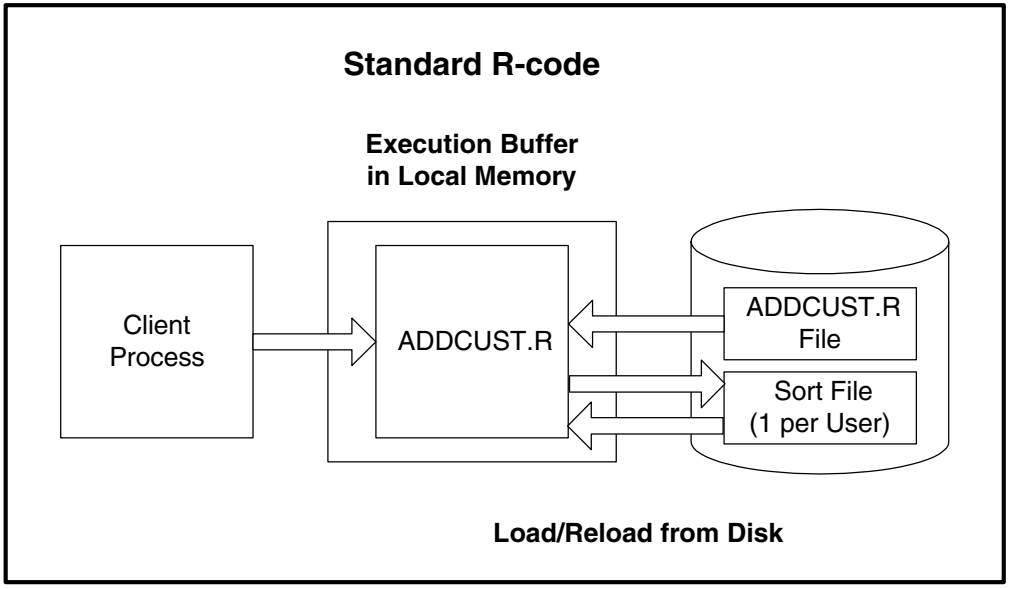



**Figure 6–1: Loading R-code from a File**

As shown in Figure 6–1, when a client process accesses an r-code procedure from an operating system file, Progress loads the procedure into local memory for that individual client and executes the segments from local memory. Progress swaps segments to the session sort file, and reloads segments from the session sort file, as necessary.

### 6.1.2 Loading R-code from a Standard Library

After locating a procedure in a standard library, Progress loads the procedure into local memory.

Figure 6–2 shows the load operation when executing an r-code procedure from a standard library.

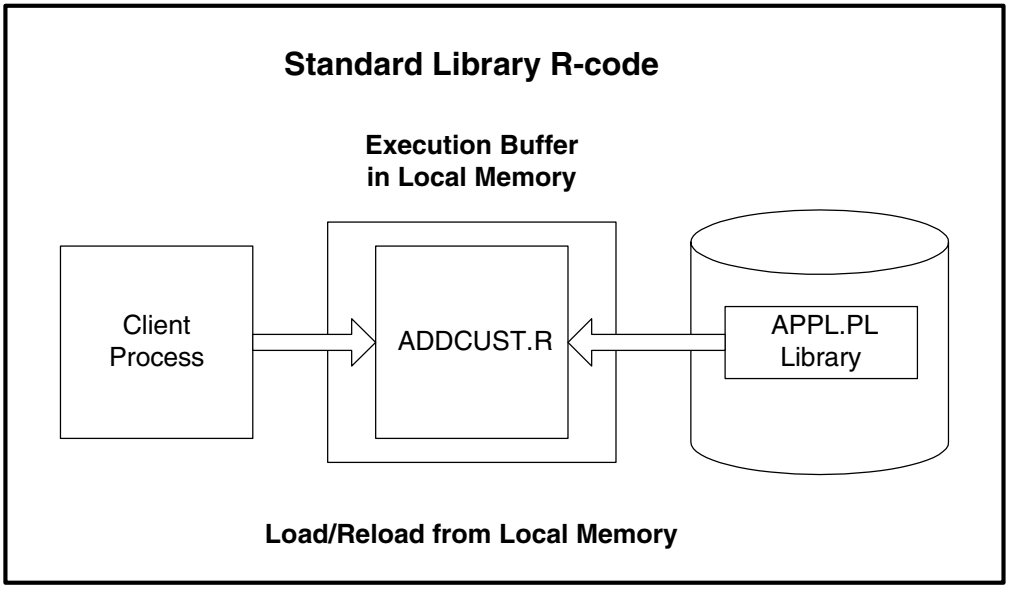



**Figure 6–2: Loading R-code from a Standard Library**

As shown in Figure 6–2, when a client process accesses an r-code procedure from a standard library, Progress loads the procedure into local memory for that individual client and executes the segments from local memory. Progress does not swap segments to and from the session sort file unless you specify the PROLIB Swap (-pls) startup parameter. By default, Progress reloads segments from the open library in local memory.

### 6.1.3 Loading R-code from a Memory-mapped Library

After locating a procedure in a memory-mapped library, Progress loads the procedure by mapping the library in shared memory where one or more clients can access it.

Figure 6–3 shows the load operation when executing an r-code procedure from a memory-mapped library.

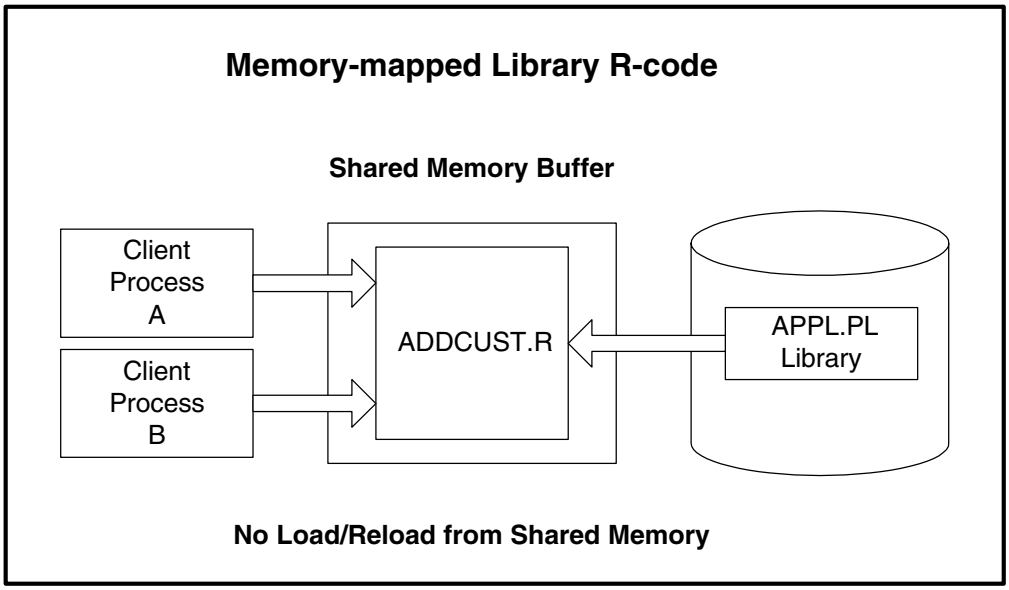



**Figure 6–3: Loading R-code from a Memory-mapped Library**

As shown in Figure 6–3, when one or more client processes access an r-code procedure in a memory-mapped library, they access the same segments in shared memory. Because Progress executes the segments from the mapped library in shared memory (not local memory), no reloading is necessary. If you specify the PROLIB Swap (-pls) startup parameter with memory-mapped libraries, Progress ignores it.

## 6.2 Setting Up a Library

Follow these steps to set up a library:

1 ♦ Create the standard library with the PROLIB utility. (All libraries must have a .pl extension.)

2 ♦ Add the r-code files to the library with the PROLIB utility.

3 ♦ If you are setting up a memory-mapped library, use the PROLIB utility to generate the memory-mapped library from the standard library.

4 ♦ Place the library in the PROPATH.

For more information about the PROLIB utility, see the "Using the PROLIB Utility" section later in this chapter. For more information about placing libraries in the PROPATH, see the "Libraries and PROPATH" section later in this chapter.

## 6.3 Running Procedures from a Library

Whether you run a procedure from an operating system file or library depends on how you reference it. You can reference a procedure in an operating system file or a library using either an absolute or relative pathname.

### 6.3.1 Using an Absolute Pathname

The most efficient but least flexible way to reference a procedure in an operating system file or a library is by using an absolute pathname. When you reference a procedure using an absolute pathname, Progress does not search the PROPATH. It looks for the procedure in the specified directory.

To run a procedure from an operating system file in a specific directory, reference the procedure using an absolute pathname. For example:

```
RUN /usr/appl/addcust.r
```

To run a procedure from a library in a specific directory, reference the library and procedure using an absolute pathname. For example:

```
RUN /usr/appl.pl<<addcust.r>>
```

When you reference a procedure in a specific library, you must enclose the member name in double angle brackets as shown.

### 6.3.2 Using a Relative Pathname

The most flexible but least efficient way to reference a procedure in an operating system file or a library is by using a relative pathname. When you reference a procedure using a relative pathname, Progress searches the PROPATH.

When you reference a procedure in either an operating system file or an unspecified library using a relative pathname, Progress searches each directory and library defined in the PROPATH until it finds the first occurrence of the procedure. For example:

```
RUN mydir/appl/addcust.r
```

When you reference a procedure in a specific library using a relative pathname, Progress searches the PROPATH until it finds the first occurrence of the library. For example:

```
RUN mydir/appl.pl<<addcust.r>>
```

When you reference a procedure in a specific library, you must enclose the member name in double angle brackets as shown.

The most flexible way to find procedures is to specify relative pathnames for all procedures run in your application. This allows you to put a procedure into a library for production, but maintain separate procedure and r-code files during development. In a development environment, either place the library in the PROPATH after your development directories or omit it from the PROPATH entirely. In a production environment, place the library at or near the beginning of the PROPATH to avoid long searches.

**NOTE:** Progress cannot find procedure files in a library; it can find only r-code files.

For more information about how libraries interact with PROPATH, see the "Libraries and PROPATH" section later in this chapter. For more information about the RUN statement, see the *Progress Language Reference*.

### 6.3.3 Related Functions

If a file is in a library, the SEARCH function returns the file's pathname using the following syntax:

```
library-pathname<<member-name>>
```

For example:

```
/usr/dictionary/dictionary.pl<<dict.r>>
```

Other Progress functions make use of this syntax. The LIBRARY function parses the pathname and returns the name of the library (in this example, `/usr/dictionary/dictionary.pl`). If the argument to LIBRARY is not in the form *`library-pathname<<member-name>>`*, the LIBRARY

function returns the unknown value (?). This is also true of the MEMBER function, which parses the pathname and returns the name of the member file (in this example, `dict.r`).

For more information about the SEARCH, LIBRARY, and MEMBER functions, see the *Progress Language Reference*.

## 6.4 Libraries and PROPATH

The following rules govern how standard and memory-mapped libraries interact with PROPATH during a Progress session:

- The first time you run a member procedure from a standard or memory-mapped library that is specified in the PROPATH, Progress locates the procedure by searching through each directory and library in the PROPATH until it finds the procedure. To search a library for a member procedure, Progress must open the library. When Progress opens a standard library, it loads the procedure into local memory. When Progress opens a memory-mapped library, it maps the library in shared memory.

- When searching through directories and libraries in the PROPATH, Progress starts at the beginning of the PROPATH and searches each directory and library, in defined order, until it finds the procedure. Thus, placing the library at the beginning of the PROPATH improves performance.

- A library remains open until the end of your Progress session or until you remove the library from the PROPATH. If you remove a library from the PROPATH while a member procedure is still active (either running, or waiting for a subprocedure to return), Progress displays an error message and the library remains open until you end your Progress session. Otherwise, Progress closes a standard library and unmaps a memory-mapped library.

- If you use a SEARCH or RUN statement to open a library that is not in the PROPATH, the library remains open until you end your Progress session.

- If you use libraries, Progress accesses r-code files as if you had specified the Quick Request (-q) startup parameter. That is, Progress searches the PROPATH only on the first reference to a procedure. Thereafter, if the procedure still resides in memory, in the local session compile file, or in an r-code library, Progress reuses the procedure instead of searching the PROPATH again. For more information about the Quick Request (-q) startup parameter, see the *Progress Startup Command and Parameter Reference*.

**NOTE:** To ensure that all users access the same memory-mapped library in shared memory, each user must place the library in their PROPATH using the same library pathname.

For more information about running procedures from a library, see the "Running Procedures from a Library" section earlier in this chapter.

## 6.5 Memory and Network Considerations

Each library has an internal directory that Progress uses to access member procedures. Progress loads this directory into memory when it opens the library. The directory stays in memory until the library is closed.

By default, Progress tries to load the entire directory of each library into memory. In some cases, you might not have enough available memory to store a library's entire internal directory. Use the PROLIB Memory (-plm) startup parameter to limit the size of a standard library's internal directory in memory. This parameter allocates a 512-byte cache for a standard library's directory. This parameter is especially useful when running in limited memory with several libraries opened simultaneously. If you specify the PROLIB Memory (-plm) startup parameter with memory-mapped libraries, Progress ignores it.

If you run Progress over a network and place a standard library on a file server, remote reads from the library might take longer than reading an r-code procedure and storing it locally in the session sort file. Use the PROLIB Swap (-pls) startup parameter to store standard library r-code procedures locally in the session sort file. If you specify the PROLIB Swap (-pls) startup parameter with memory-mapped libraries, Progress ignores it.

For more information about the PROLIB Memory (-plm) and PROLIB Swap (-pls) startup parameters, see the *Progress Startup Command and Parameter Reference*.

## 6.6 Using the PROLIB Utility

The PROLIB utility allows you to create and maintain standard libraries and generate memory-mapped libraries. This is the syntax to start the PROLIB utility:

| Operating System | Syntax |
|---|---|
| UNIX Windows | `prolib` *library-name parameter* [ *file-name* ... ] [ *parameter* [ *file-name* ... ] ... ] |

*library-name*

Specifies the name of an r-code library. The library name must have a .pl extension.

*parameter*

Specifies what action to take on the library. Table 6–1 lists the parameters and their descriptions.

**Table 6–1: PROLIB Utility Parameters**

| Parameters | Description |
|---|---|
| -create | Creates a new standard library. |
| -makeshared | Generates a memory-mapped library from a standard library. |
| -add | Adds files to a standard library. |
| -replace | Replaces files in a standard library. |
| -delete | Deletes files from a standard library. |
| -list | Lists library information such as library name, format, code page, and file contents. |
| -extract | Extracts files from a standard library. This parameter copies a file from the library into another file, outside of the library, and gives it the same name. |
| -yank | Extracts files from a standard library and places them in the current working directory. |
| -compress | Compresses a standard library by making a copy of it. |
| -nowarn | Suppresses any warning message that might occur during the operation of the primary parameters. If you add a file to a standard library with the -add and -nowarn parameters, and the file already exists in the library, PROLIB replaces the file. |
| -pf | Allows you to supply command-line arguments in a parameter file. You cannot use -pf in a parameter file. |
| -verbose | Directs PROLIB to display processing information that is ordinarily suppressed. |
| -date | Specifies the format of the date as it appears when you use the -list parameter. |

When specifying a parameter, you do not have to type the complete parameter name. You can type the minimally unique string for each parameter (for example, -l for -list and -e for -extract).

The -nowarn and -verbose parameters modify the behavior of the -create, -makeshared, -add, -replace, -delete, -list, -extract, and -yank parameters.

You can place the -nowarn, -pf, and -verbose parameters anywhere on the command line. They affect the processing of all other specified parameters.

You must place the -create parameter before all other parameters. PROLIB processes parameters in left-to-right order as they appear on the command line. If an error occurs during the PROLIB command, PROLIB terminates. This behavior occurs so that options specified later in the command line, which might depend on the failed option, do not execute.

You cannot use the -add, -replace, -delete, -extract, -yank, or -compress parameters with a memory-mapped library (that is, when you specify a memory-mapped library in *library-name*).

*file-name*

Specifies the name of an r-code file, or a memory-mapped library file when using the -makeshared parameter.

### 6.6.1 Using Wild Cards

The -add, -replace, -extract, -list, -delete, and -yank parameters accept wild card arguments. Wild card arguments can contain only the ? or * wild card characters. Depending on the parameter you want to use, you must specify these arguments in one of two ways:

- Using your operating system's regular wild card conventions
- Escaping the wild cards

The -add and -replace parameters act on operating system files. PROLIB uses system calls to copy the files. Thus, when you use these parameters, use your operating system's standard conventions to specify wild cards.

The -extract, -list, -delete, and -yank parameters act on files that are already placed in a library. PROLIB does not use system calls to act on the files, but instead uses its own internal code. On UNIX, you must escape your wild card arguments by either enclosing them in quotes (for example, `prolib app.pl -delete "sys*.r"`) or escaping the wild card characters individually (for example, `prolib app.pl -delete sys\*.r`). Your operating system might use different

techniques to escape wild cards. In Windows, you do not need to escape wild card arguments because the operating system does not expand them before passing them to PROLIB.

### 6.6.2 Creating a Standard Library

To create a standard library, enter the following command:

| Operating System | Syntax |
|---|---|
| UNIX<br>Windows | `prolib` *`library-name`*<br>`-create [ -codepage` *`codepage-name`* `]` |

*`library-name`*

Specifies the name of the library you want to create.

All libraries must have a .pl extension. You can add libraries to the PROPATH environment variable by specifying the file's absolute or relative pathname.

*`codepage-name`*

Specifies the name of the code page for the library you want to create.

A library can contain r-code procedures with different code pages. All r-code procedures retain their respective code pages.

If you do not specify the -codepage parameter at creation time, Progress marks the r-code library with the code page specified in the Internal Code Page (-cpinternal) startup parameter.

For information about code page compatibility, see the section "Code Page Compatibility" later in this chapter.

### 6.6.3 Generating a Memory-mapped Library

To generate a memory-mapped library from a standard library, enter the following command:

| Operating System | Syntax |
|---|---|
| UNIX Windows | `prolib` *standard-library-name* `-makeshared` *mapped-library-name* |

*standard-library-name*

Specifies the name of the standard library from which to generate the memory-mapped library. The library name must have a .pl extension. You can add libraries to the PROPATH environment variable by specifying the file's absolute or relative pathname.

**NOTE:** Save the standard library. It is the source of modification for the memory-mapped library. Each time you modify the standard library you must regenerate the memory-mapped library.

*mapped-library-name*

Specifies the name of the memory-mapped library you are generating. The library name must have a .pl extension.

**NOTE:** You cannot modify a memory-mapped library directly. To modify a memory-mapped library, you must modify its source standard library and regenerate the memory-mapped library.

PROLIB generates a memory-mapped library with the standard library's code page. All text segments in all r-code procedures retain their respective code pages.

For information about code page compatibility, see the section "Code Page Compatibility" later in this chapter.

### 6.6.4 Adding Files to a Standard Library

To add files to a standard library, enter the following command:

| Operating System | Syntax |
|---|---|
| UNIX Windows | `prolib` *library-name* `-add` *file-name* [ *file-name* ] ... |

*library-name*

Specifies the name of the library where you want to add the file.

*file-name*

Specifies the pathname of the file. You can add a file to a library by specifying the file's absolute or relative pathname.

When you add a file to a library, the library entry for that file retains the entire pathname. For example, enter a file into a library called `newlib.pl` using the following command:

```
prolib newlib.pl -add /usr/apps/proc1.r
```

The pathname `/usr/apps/proc1.r` appears in the library's table of contents. You can display the library's table of contents with the -list parameter.

When you extract a file from a library, PROLIB copies the file into the directory specified by the pathname (in this example, `/usr/apps`). See the "Extracting Files from a Standard Library" section for more information.

If you try to add a file that already exists, PROLIB displays an error message. However, if you specify the -nowarn parameter, PROLIB replaces the existing file. See the "Replacing Files in a Standard Library" section for more information.

### 6.6.5 Replacing Files in a Standard Library

To replace one or more files in a standard library, enter the following command:

| Operating System | Syntax |
|---|---|
| UNIX<br>Windows | prolib *library-name*<br>-replace *file-name* [ *file-name* ] ... |

*library-name*

Specifies the name of the library with the file you want to replace.

*file-name*

Specifies the name of the file.

When you replace a file, the system overwrites the old file with a file of the same name. Therefore, the pathname you supply for the file has to be the same as the pathname you used originally to add the file to the library. If you try to replace a file that does not exist, PROLIB displays an error message. However, if you specify the -nowarn parameter, PROLIB adds the existing file to the library. See the "Adding Files to a Standard Library" section for more information.

### 6.6.6 Deleting Files from a Standard Library

To delete files from a standard library, enter the following command:

| Operating System | Syntax |
|---|---|
| UNIX<br>Windows | prolib *library-name*<br>-delete *file-name* [ *file-name* ] ... |

*library-name*

Specifies the name of the library with the file you want to delete.

*file-name*

Specifies the name of the file. You can specify more than one file at a time.

### 6.6.7 Listing the Contents of a Library

To list the contents of a library, enter the following command:

| Operating System | Syntax |
|---|---|
| UNIX Windows | `prolib` *`library-name`*<br>`-list [` *`file-name`* `] ...` |

*`library-name`*

Specifies the name of the library with the contents you want to list.

*`file-name`*

Specifies that PROLIB should list information about that file only; *file-name* has to be the same absolute or relative pathname as when you originally add the file. You can specify more than one file at a time.

Table 6–2 lists the information that appears in the header section of the output.

**Table 6–2: List (-list) Parameter Library Information**

| Information | Description |
|---|---|
| Library Name | The name of the library. |
| Code Page | The code page in which the library was created. |
| Format | The format of the library. The valid library formats are STANDARD and MAPPED. |

Table 6–3 lists the information that appears in the output for each file in the library.

**Table 6–3: List (-list) Parameter File Information** *(1 of 2)*

| Information | Description |
|---|---|
| Name | The name of the file. |
| Size | The size of the file in the library, in bytes. |

**Table 6–3: List (-list) Parameter File Information** *(2 of 2)*

| Information | Description |
|---|---|
| Type | The file type. PROLIB recognizes two file types: R (r-code file type) and O (any other file type). Although libraries are specifically designed to hold r-code files, you can place any type of file in them. |
| Offset | The distance, in bytes, of the start of the file from the beginning of the library. |
| Modified | The date and time of the file's last modification. |
| Added To Lib | The date and time the file was entered into the library.<br><br>You can also change the format of the dates as they appear in the Modified and Added To Lib fields with the -date parameter, where *date-format* is three letters: m (month), d (day), and y (year). To specify a format, place the letters in the order you want the months, days, and years to appear on screen. The default is mdy. |

### 6.6.8 Extracting Files from a Standard Library

To extract files from a standard library, enter the following command:

| Operating System | Syntax |
|---|---|
| **UNIX Windows** | `prolib` *`library-name`*<br>`-extract` *`file-name`* **[** *`file-name`* **]** **...** |

*library-name*

Specifies the name of the library with the file you want to extract.

*file-name*

Specifies the name of the file. You can specify more than one file at a time.

When you extract a file from a library, PROLIB copies the file and places it in its original directory. Depending on how you add a file to a library, PROLIB extracts differently:

- If you added the file by specifying a relative pathname, PROLIB searches your current working directory for the subdirectory indicated in the pathname. For example, if you added a file by using the relative pathname `apps/proc1.r`, PROLIB searches your current working directory for the subdirectory `apps`. If the directory does not exist, PROLIB displays an error message.
- If you added the file by specifying its absolute pathname, PROLIB searches for the original directory. If the directory does not exist, PROLIB displays an error message.

**NOTE:** If you specify the -yank parameter instead of -extract, Progress strips everything but the filename from the pathname and places the file in your current directory.

If the file you are extracting (or yanking) already exists in the directory outside of the library, PROLIB displays the following message:

```
File file-name already exists. Do you want to replace it?
Please answer 'yes' or 'no' or 'rename' to assign new name:
```

where *file-name* is the pathname of the file you are extracting (or yanking). If you answer yes, PROLIB overwrites the file. If you answer no, PROLIB does not overwrite the file. If you answer rename, PROLIB displays the following message:

```
Enter new name to use for extracted file file-name -->
```

where *file-name* is the pathname of the file you are extracting (or yanking). After you enter the pathname, PROLIB gives the file the new pathname and places it in the appropriate directory.

### 6.6.9 Extracting Files to Your Current Directory

When extracting a file with the -yank parameter, PROLIB discards everything but the filename from the file's pathname. For example, if you added a file using the pathname `/usr/apps/proc1.r`, PROLIB discards `/usr/apps` from the pathname and places `proc1.r` in your current working directory.

To extract files from a library using the -yank parameter, enter the following command:

| Operating System | Syntax |
|---|---|
| UNIX<br>Windows | prolib *library-name*<br>-yank *file-name* [ *file-name* ] ... |

*library-name*

Specifies the name of the library with the file you want to extract.

*file-name*

Specifies the name of the file. You can specify more than one file at a time.

### 6.6.10 Compressing a Standard Library

To compress a standard library, enter the following command:

| Operating System | Syntax |
|---|---|
| UNIX<br>Windows | prolib *library-name* -compress |

*library-name*

Specifies the name of the library you are compressing. The -compress parameter removes extra spaces that occur in the library as a result of repeated adds or deletes.

To compress a library, PROLIB creates a temporary file that requires an area of disk space equal to the size of the compressed library. Before compressing a library, make sure you have enough disk space for the temporary file. Temporary files have names that begin with the letters PLB, followed by numbers and possibly letters.

If your system goes down during a compress operation, the temporary file might remain on your disk. Delete this file to free up disk space. The original library will not be damaged.

Repeated adds create empty spaces in the library over time. To minimize the rate at which these empty spaces are created, you can use one command instead of many. For example, instead of entering the following commands:

```
prolib test.pl -add test.r
prolib test.pl -add test2.r
prolib test.pl -add test3.r
```

enter this command:

```
prolib test.pl -add test*.r
```

You can use the -list parameter to determine how much space in your library is unused.

### 6.6.11 PROLIB Command Examples

This section provides several PROLIB command examples.

The following example creates a standard library called `app.pl` in your current directory:

```
prolib app.pl -create
```

The following example adds all the r-code files in the current directory that begin with the letters sys to the standard library `app.pl`. In this example, the r-code files are in a directory, so the operating system processes the wild card argument:

```
prolib app.pl -add sys*.r
```

The following example deletes all r-code files ending with the letters dev from the standard library `app.pl` and lists all of the files in the library. In this example, the r-code files are in the `app.pl` library, so PROLIB processes the wild card argument. This requires escaping the `*dev.r` argument. On UNIX, this means placing quotes around it. Your operating system might use different techniques to escape wild cards:

```
prolib app.pl -delete "*dev.r" -list
```

The following example replaces the files `appmenu.r` and `apprept.r` in the standard library `app.pl`, then compresses the library to free up unused disk space:

```
prolib app.pl -replace appmenu.r apprept.r -compress
```

The following example creates a standard library called `app.pl` in your current directory, adds all the r-code files in the current directory that begin with the letters sys to the library `app.pl`, and generates a memory-mapped library called `app_map.pl`:

```
prolib app.pl -create -add sys*.r -makeshared app_map.pl
```

## 6.7 Code Page Compatibility

The Internal Code Page (-cpinternal) startup parameter determines the internal code page that Progress uses in memory. The code page in which r-code is compiled and the internal code page of the session in which the r-code runs must be compatible. That is, the r-code code page must be either the internal code page, undefined, or convertible to the internal code page.

If the code page for a standard r-code procedure, from either an operating system file or a standard library, is not the same as the session's internal code page, Progress performs an in-memory conversion of the r-code text segments to the session's internal code page.

**NOTE:** If you use the R-code In Code Page (-cprcodein) startup parameter to specify a code page for reading r-code text segments, Progress reads the text segments as if they were written in that code page, even if the text segments were written in a different code page.

Since r-code procedures in memory-mapped procedure libraries are read-only, Progress cannot perform an in-memory code page conversion of r-code text segments. If the code page of a memory-mapped r-code procedure is not compatible with the internal code page of the session in which the r-code runs, Progress will not execute the r-code. It will report an error and stop execution.

For more information about the Internal Code Page (-cpinternal) and R-code In Code Page (-cprcodein) startup parameters, see the *Progress Startup Command and Parameter Reference*.

# 7

# Managing Print Devices

System administrators are responsible for setting up and maintaining the output devices required by an application. Most Progress applications access printers to print reports or other output. This chapter provides an overview of how to set up print devices on Progress-supported operating systems. It also describes how to use Progress to set up and control output routing and printing characteristics.

This chapter provides information about the following topics:


- "Printing in Progress"
- "Printing on UNIX"
- "Printing in Windows"
- "Setting Up Output-routing"
- "Configuring a Printer"

## 7.1 Printing in Progress

Most operating systems support two general printing techniques: direct printing and spooled printing. *Direct printing* sends application output directly to a printer set up on the operating system. This technique of printing is frequently found on single-user systems. However, you can use direct printing in other situations (for example, printing tickets and special forms).

*Spooled printing* sends application output to a temporary file and places a print request in a print queue set up on the operating system. Spooling permits you to queue a print request and continue processing; the system executes the print request when a printer becomes available. A spooled device can help balance the demand on line printers if you are running applications on a time-shared system.

Progress runs with most printers and, depending on the operating system in use, makes default assumptions about printer setup. Progress uses the printer port or spooler named at execution time, not at compile time. Thus, precompiled Progress applications work regardless of the printer port or spooler in effect.

This section provides an overview of the Progress language elements you can use to direct application output, which includes the:

- OUTPUT TO statement
- OUTPUT THROUGH statement

### 7.1.1 OUTPUT TO Statement

By default, all Progress output is directed to the terminal. The OUTPUT TO statement allows an application to redirect output to other output devices. Following is a partial syntax example of the OUTPUT TO statement:

```
OUTPUT [ STREAM stream ] TO
  {     PRINTER [ printer-name ]
      | opsys-file
      | opsys-device
      | TERMINAL
      | VALUE ( expression )
      | "CLIPBOARD"
  }
        .
        .
        .
  [ PAGED ]
  [ PAGE-SIZE { constant | VALUE ( expression ) } ]
        .
        .
        .
```

Use the OUTPUT TO statement with the PRINTER option to send output to the default print device on the current operating system. For example:

```
OUTPUT TO PRINTER.
```

On UNIX, the OUTPUT TO PRINTER statement sends output to the default spooler. To use a spooler other than the default, start Progress using the Printer (-o) startup parameter.

In Windows, the OUTPUT TO PRINTER statement sends output to the printer defined in default print context. The default print context is the set of values that defines the default printer and setup for that printer in Windows. If there is no default print context, Progress uses the printer control settings from the current environment.

Use the SYSTEM-DIALOG PRINTER-SETUP statement to display the Windows Print dialog box and let the user change the default print context at runtime. Use the PRINTER-NAME attribute of the SESSION system handle to change the printer name in the default print context without user intervention. You can change the system default printer from the Windows Control Panel.

Use the OUTPUT TO PRINTER statement with its various options to override the default print context for a specific print job. For example, send output to a printer other than the default printer by using the following syntax:

```
OUTPUT [ STREAM stream ] TO PRINTER printer-name
```

Progress assumes that all printers can handle hard-coded form feed characters (**CTRL-L** or 0C Hex). The OUTPUT TO PRINTER statement automatically paginates Progress output on the default printer. To paginate output on a printer, other than the default printer, use the OUTPUT TO PRINTER statement with the PAGED or PAGE-SIZE option.

Use the Printer (-o) startup parameter to specify a default printer to use when processing the OUTPUT TO PRINTER statement in procedures. The specified printer must be set up on the current operating system.

For a complete description of the OUTPUT TO PRINTER statement, the SYSTEM-DIALOG PRINTER-SETUP statement, and the SESSION system handle, see the *Progress Language Reference*. For more information about the Printer (-o) startup parameter, see the *Progress Startup Command and Parameter Reference*.

### 7.1.2 OUTPUT THROUGH Statement

The OUTPUT THROUGH statement allows an application to redirect output to a process that Progress starts. On UNIX, this statement allows you to pipe Progress output to a UNIX process. Following is a partial syntax example of the OUTPUT THROUGH statement:

```
OUTPUT [ STREAM stream ] THROUGH
 { program-name | VALUE ( expression ) }
 [ argument | VALUE ( expression ) ] ...
  .
  .
  .
```

You can also use the OUTPUT THROUGH statement to access spooling capabilities on UNIX systems. For example, this command sends output to the printer named lw on a UNIX system:

```
OUTPUT THROUGH lpr -Plw.
```

For a complete description of the OUTPUT THROUGH statement, see the *Progress Language Reference*.

## 7.2 Printing on UNIX

Since UNIX is a multi-tasking operating system, print devices are usually spooled. However, it is possible to print directly to a device that is not spooled. This section describes the setup of spooled print devices on UNIX System V and Berkeley 4.*n* systems. The default spooler name for all systems that run UNIX System V is `lp`. The default spooler name for all other UNIX systems, including Berkeley 4.*n*, is `lpr`.

To set up a printer on a UNIX System V system, make sure that the `lp` system and the printer hardware are installed and that the LP scheduler is shut down. The *LP scheduler* services print requests for all destinations by routing requests to the interface programs that do the printing on devices. The LP scheduler is usually started when you boot your system. To shut down the LP scheduler, use the `lpshut` command. Log in as the root user, then follow these steps:

1 ♦ Install and configure a device in the `/dev` directory. Make sure the device is not configured as a login terminal.

2 ♦ Use the `lpadmin` command to set up a new printer. This command allows you to specify a printer name and a printer-interface program, and connects them with a print device that exists in the `/dev` directory.

3 ♦ Use the `lpsched` command to start the LP scheduler once the printer has been set up.

4 ♦ Use the `accept` and `enable` commands to let the printer accept print requests and enable spooling for the printer.

To set up a printer on a UNIX Berkeley 4.*n* system, make sure that the `lpr` system and the printer hardware are installed and that the lpd daemon is running. The *lpd daemon* is the master server that services print requests and routes them to the proper spooling directories for printing. The lpd daemon usually starts when you boot your system. Log in as the root user, then follow these steps:

1 ♦ Install and configure a device in the `/dev` directory. Make sure the device is not configured as a login terminal.

2 ♦ Edit the `/etc/printcap` file. This file contains definitions for all of the printers on a system (or accessible over a network). Each printer definition in `/etc/printcap` does the following:

- Designates a device in the `/dev` directory as the printer port
- Designates a spooling directory
- Sets up printer characteristics

3 ♦ Create the spooling directory and make sure the spooling directory and the directories of the lpr system have the proper permissions and ownership.

4 ♦ Use the `lpc start` and `lpc enable` commands to let the printer accept print requests and enable spooling for the printer.

Most UNIX System V or Berkeley 4.*n* systems have printer installation scripts that perform the printer installation steps described in this section.

The OUTPUT TO PRINTER statement sends output to the UNIX system default spooler. To use a spooler other than the default, start Progress using the Printer (-o) startup parameter followed by the name of the spooler. For example:

```
pro database-name -o "lp -dpx1"
```

Alternately, you can use the OUTPUT THROUGH statement to direct output to a spooler other than the default spooler:

```
OUTPUT THROUGH lp spooler-options PAGED.
```

or:

```
OUTPUT THROUGH lpr spooler-options PAGED.
```

With both the OUTPUT THROUGH statement and the Printer (-o) startup parameter, you can specify UNIX spooler options for the spooler that you designate. If you want to specify spooler options with the Printer (-o) startup parameter, surround the spooler name and options with quotes. For example:

```
pro database-name -o "lpr -Plw"
```

For more information about printing on UNIX, see the UNIX system documentation.

### 7.2.1 UNIX Networks and Printing

UNIX systems can be integrated into a variety of networks. This section describes how to set up remote printers using the TCP/IP networks and the Basic Networking Utilities. You can use both of these protocols to connect UNIX System V and Berkeley 4.*n* systems.

On TCP/IP networks, remote spooling is handled with two spooling queues, one on the local machine and one on the remote machine.

Follow these steps to set up remote printers using TCP/IP networks:

1 ♦ Be sure that the required TCP/IP system files are set up on both the local and remote machines. The local and remote machines have entries in their `/etc/hosts` and `/etc/hosts.equiv` files to facilitate network operations.

2 ♦ Be sure that the printer is set up on the remote machine.

3 ♦ Define an entry for the remote printer in the `/etc/printcap` file on the local machine. This remote-printer entry must have an empty field for the lp capability, a remote machine designation, a remote printer designation, and a remote spooler directory designation. The remote printer that you define in the `/etc/printcap` file can be the default print spooler for the local machine.

4 ♦ Create the spooling directory, then use the lpc start and lpc enable commands to allow the remote printer to receive print requests and to enable spooling for the printer.

When you initiate a remote print job with lpr, the job is queued locally and a daemon process is created to oversee the transfer of the job to the remote machine. If the destination machine is unreachable, the job remains queued until it is possible to transfer the files to the spooling queue on the remote machine.

Follow these steps to set up remote print spooling using the Basic Networking Utilities:

1 ♦ Be sure that the Basic Networking Utilities (uucp) are installed on both the remote and local machines. The Basic Networking Utilities use a series of files to define network machines (`L.sys`), to set up communications mechanisms and characteristics (`L.devices` and `L-dialcodes`), to determine what commands can be executed remotely (`L-cmd`), and to set up network permissions (`USERFILE`). An entry is also required in the `/etc/passwd` file on each network machine to allow uucp logins.

2 ♦ Be sure that the printer is set up on the remote machine.

3 ♦ Create an executable script to use the uux command to enact a print job on the printer attached to the remote node.

When you use the uux command to execute a remote print job, the network request is queued in the local uucp spooling directory, permissions are checked, and the request is sent to the remote machine. The remote machine places the print request in the spooling directory of the printer.

For more information about UNIX networks and printers, see the UNIX system administration documentation.

## 7.3 Printing in Windows

In Windows, you can choose either of two printing modes:

- Default mode
- Windows Passthrough Printing (WPP)

In default mode, Progress uses the Windows device drivers. This mode is sufficient for all basic printing that does not use any printer access control codes, such as those sent by PUT, PUT CONTROL, or DISPLAY statements.

In WPP mode, Progress bypasses the Windows device drivers. This means you can send control codes (such as POSTSCRIPT commands) directly to the printer with the PUT, PUT CONTROL, or DISPLAY statements. To enable windows passthrough printing for a session, use the Windows Passthrough Printing (-wpp) startup parameter. You can only use this parameter on the command line.

You cannot use WPP mode with a POSTSCRIPT printer to print Progress standard reports, such as those produced by the Data Dictionary.

**NOTE:** When using windows passthrough printing under Windows95, you must change the Spool Data Format from EMF spooling to RAW spooling. Use the Control Panel to access the Printer Properties for the printer, then select Spool Setting from the Details tab of the Printer Properties tab. Change the Spool Data Format to RAW spooling.

For a complete description of the PUT, PUT CONTROL, and DISPLAY statements, see the *Progress Language Reference*. For more information about the Windows Passthrough Printing (-wpp) startup parameter, see the *Progress Startup Command and Parameter Reference*.

## 7.4 Setting Up Output-routing

For systems with multiple printers, you might want to set up an output-routing scheme. Output-routing schemes direct application output to different devices based on criteria that you specify. Since Progress uses the printer port or spooler named at execution time, not at compile time, you can set up output-routing schemes to send application output to different printers based on user identification, the type of procedure the user runs, or some other criteria.

When setting up an output-routing scheme, examine the application and system requirements. For example, your system users might want to direct output to different printers. In this case, you would have to set up an output-routing scheme based on the user identification. An application might have reports that require different types of printers. For example, one report might require a 132-character printer and another report might require an 80-character printer. In this case, you would have to set up an output-routing scheme based on the report type.

Another output-routing scheme could direct output to different printers based on both the user and the type of report.

You can set up an output-routing scheme in the following ways:

- Using a Startup Procedure
- Using an Output-routing Table
- Using Interactive Output-routing

### 7.4.1 Using a Startup Procedure

The simplest output-routing scheme consists of setting up a Progress application startup procedure for each printer on a system. Each startup procedure could contain the Printer (-o) startup parameter to designate a particular printer as the default printer for the current Progress session.

Figure 7–1 shows a sample application startup procedure on UNIX. This application startup procedure designates lpr -Plw as the default print device for the current Progress session and starts up a Progress application located in the `/usr/appl1` directory that access a database called appldb.

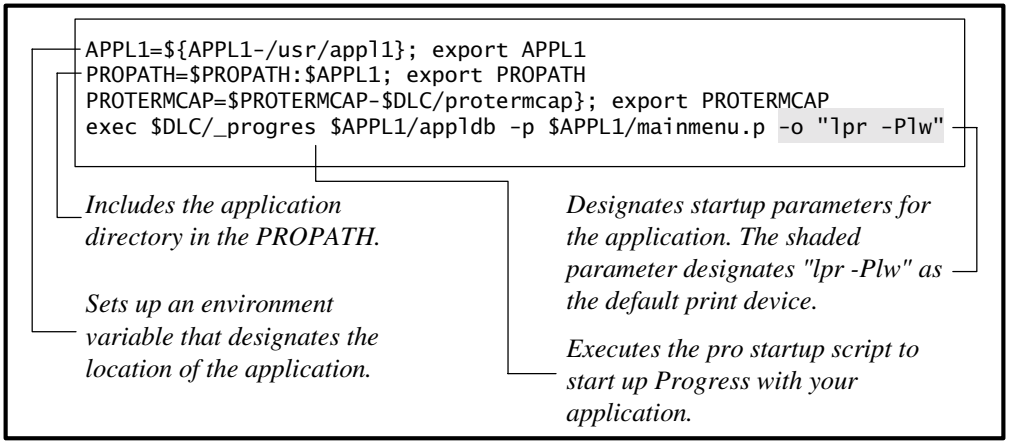



**Figure 7–1: Sample Application Startup Procedure (UNIX)**

Figure 7–2 shows a sample application startup procedures on NT. This application startup procedure designates LPT2 as the default print device for the current Progress session and starts up a Progress application located in the /usr/appl1 directory that access a database called appldb.

```
echo off
set APPL1=\usr\APPL1
set PROPATH=%PROPATH%;%APPL1%
%DLC%\_progress -D 20  -o LPT2 -p
\usr\APPL1\mainmenu.p appldb
```

**Figure 7–2: Sample Application Startup Procedure (NT)**

After the startup procedures are set up for each system printer, you can designate users to use different startup procedures depending on which output destination they want. For easy user access and maintenance, all of the startup procedures should be located in the directory where the system software is located.

### 7.4.2 Using an Output-routing Table

More complex output-routing schemes, such as routing output based on the user ID, require the creation of several related Progress tables and an include file that can access these tables and select the proper output command for the particular user.

Just as you can use Progress database tables (_User) to set up security for Progress applications, you can also use Progress tables for output routing. If you want to route application output to different printers based on user ID, set up a Progress table in the application database containing each of the valid user IDs and the corresponding print commands.

Figure 7–3 shows a sample output-routing table.

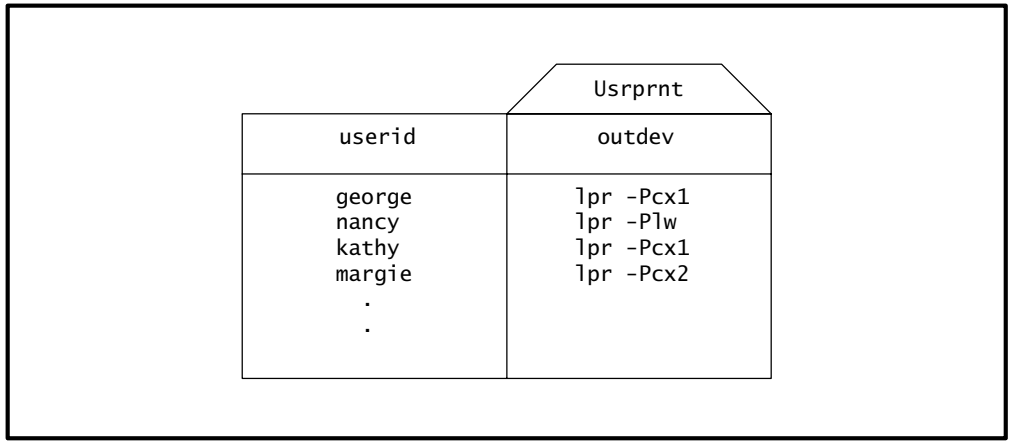



**Figure 7–3: Sample Output-routing Table**

The database table shown in the figure contains records that have a userid field and an outdev field containing an operating system output device. The userid field is a unique primary index for the table. You have to keep the Usrprnt table up to date with a record for each new user on your system.

After setting up the database table, create a Progress include file that determines the current user ID and then searches the Usrprnt table for a matching user ID and corresponding output device. For example:

```
FIND Usrprnt WHERE Usrprnt.userid = USERID NO-ERROR.
  IF AVAILABLE Usrprnt THEN
    OUTPUT THROUGH VALUE(Usrprnt.outdev).
  ELSE
    OUTPUT TO PRINTER.
```

The include file searches the Usrprnt table for a record that has a user ID that matches the user ID established for the current application session. If there is no user ID match or established user ID for the application session, Progress uses the default system printer. If there is a match, Progress sends the application output to the output device contained in the corresponding outdev field.

The VALUE option of the OUTPUT THROUGH statement allows you to designate a substitutable output device. The USERID function allows you to capture system user IDs on UNIX. For more information about the _User table, see *Progress Database Administration Guide and Reference*.

After creating an include file to route application output, replace all OUTPUT statements in your application with the include file. For example:

```
{output1.i}.

FOR EACH customer:
  PUT cust-num name address city st zip.
END.
```

### 7.4.3 Using Interactive Output-routing

You can also set up interactive output-routing schemes that let application users select an output device at run time. In Windows, for example, you can change the destination of the OUTPUT TO PRINTER statement at run time by using the SYSTEM-DIALOG PRINTER-SETUP statement.

## 7.5 Configuring a Printer

Many printers have a set of control sequences that you can use to specify different print characteristics, such as compressed printing or italics typeface. These control sequences often involve special characters that can be represented by their octal (base 8) equivalent. To denote these octal codes, precede the three octal digits with an escape character. On UNIX, the escape character is a tilde (~) or a backslash (\). Table 7–1 lists the standard control sequences using a tilde as the escape character.

**Table 7–1: Standard Printer Control Sequences** *(1 of 2)*

| Sequence | Interpreted as | Comments |
|---|---|---|
| ~*nnn* | Single character | Use to send a single character to the printer, where *nnn* is an octal code between 000 and 377. You must specify all three digits. |
| ~t | Tab character | Use to send the 011 octal code to the printer. |
| ~r | Carriage return | Use to send the 015 octal code to the printer. |
| ~n | Line feed | Use to send the 012 octal code to the printer. |
| ~E | Escape | Use to send the 033 octal code to the printer. |
| ~b | Backspace | Use to send the 010 octal code to the printer. |

**Table 7–1: Standard Printer Control Sequences** *(2 of 2)*

| Sequence | Interpreted as | Comments |
|---|---|---|
| ~f | Form feed | Use to send the 014 octal code to the printer. |
| ~" | " | Use within quoted strings as an alternative to two quotes (""). |
| ~\ | \ | Use to send the backslash character to the printer. |
| ~~ | ~ | Use to send the tilde character to the printer. |

The PUT statement with the CONTROL option allows you to specify a control sequence that you want to send to a printer. The PUT statement has the following syntax:

```
PUT [ STREAM stream-name ]
  CONTROL "ctrl-sequence" "ctrl-sequence" ...
```

In the following example, the control sequence ~017 turns on the compressed printing feature of the current print device:

```
PUT STREAM a CONTROL "~017".
```

The control sequences sent to the output destination do not affect the current line, page counters, and positions maintained within Progress. For more information about the PUT CONTROL statement, see the *Progress Language Reference*.

### 7.5.1 Creating a Printer-control Table

It is important to note that control sequences are hardware dependent. Applications with hard-coded printer-control sequences are not easily transportable. You cannot change the print devices that the application accesses without changing the hard-coded printer-control sequences in the application. To bypass this obstacle, create a printer-control table. The printer-control table contains a record for each printer on your system. Each record in the printer-control table must contain a field for the name of the print device and several other fields containing a corresponding set of standard control sequences.

There is no easy way to employ a Progress UPDATE statement to interpret the printer-control sequences and enter them into a printer-control table. For example, the Progress UPDATE statement interprets the octal sequence ~017 as the character string:

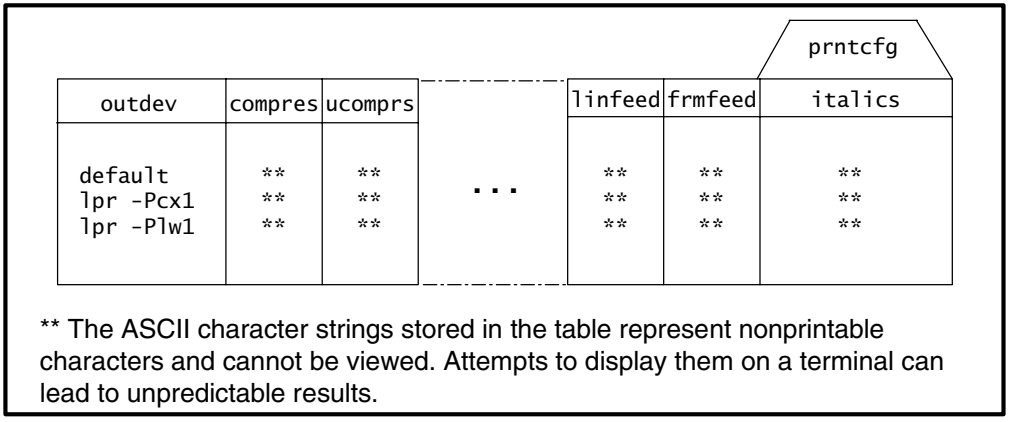
```
[TILDE][ZERO][ONE][SEVEN]
```

instead of the proper code sequence:

```
[CONTROL-O]
```

You can create a system administration procedure that lets you convert printer-control sequences into the appropriate ASCII character strings, and store the printer-control sequences as ASCII character strings in the fields of a printer-control table.

Keep the printer-control table up to date with a record for each printer on your system. Figure 7–4 shows a sample printer-control table.




** The ASCII character strings stored in the table represent nonprintable characters and cannot be viewed. Attempts to display them on a terminal can lead to unpredictable results.

**Figure 7–4: Sample Printer-control Table**

The field containing the print device name must index the table. In this sample control table, the outdev field indexes the table.

To access the proper control sequences for a printer from a printer-control table, set up an include file to determine the print device used in the application, then substitute the corresponding control sequences into the PUT CONTROL statements of the application. For example:

```
FIND Usrprnt WHERE Usrprnt.userid = USERID NO-ERROR.
  IF AVAILABLE Usrprnt
  THEN DO:
    OUTPUT THROUGH VALUE(Usrprnt.outdev).
    FIND Prntcfg WHERE Prntcfg.outdev = Usrprnt.outdev NO-ERROR.
  END.
  ELSE DO:
    OUTPUT TO PRINTER.
    FIND Prntcfg WHERE Prntcfg.outdev = "default".
  END.
```

The include file searches the Usrprnt table to determine the output device (`Usrprnt.outdev`) for the current user ID. After determining the output device, the include file searches the printer-control table to find the appropriate control sequences for the current printer. Notice that the printer-control table contains a default entry for the default printer.

The PUT CONTROL statements in your application should use a field in the printer-control table or a variable name to access the proper control sequence for the current printer. This allows you to change print devices without recoding the printer-control sequences for all the PUT CONTROL statements in your application. For example:

```
{output2.i}.

PUT CONTROL Prntcfg.compres.
FOR EACH customer:
  PUT cust-num name address city st zip.
END.
PUT CONTROL Prntcfg.ucomprs.
```

In this procedure, the first PUT CONTROL statement turns on compressed printing, and the second PUT CONTROL statement turns off compressed printing for the current print device.

# A

# Building Progress Executables

When you purchase a Progress product, you receive a set of executable files, database files, and other files that are necessary to run the product on your system. In most cases, the files included in your initial product installation fulfill your system requirements. However, you might need to customize a Progress executable by adding or removing product capabilities. For example, you might need to change network protocols or access a non-Progress data source. When you need to customize a Progress executable, use the PROBUILD utility. PROBUILD is an end-user configuration utility that helps you to build customized versions of certain Progress executables.

This appendix provides information about the process and requirements for building customized Progress executables. It also provides instructions for using PROBUILD on UNIX and in Windows. This appendix contains the following sections:


- "The Build Process"
- "Software and Hardware Requirements"
- "Licensing Requirements"
- "PROBUILD Overview"
- "Building Executables on UNIX"
- "Building Executables in Windows"
- "Troubleshooting"

## A.1 The Build Process

The process for building customized executables consists of six basic steps.

Figure A–1 illustrates the steps you must follow to build a customized executable.

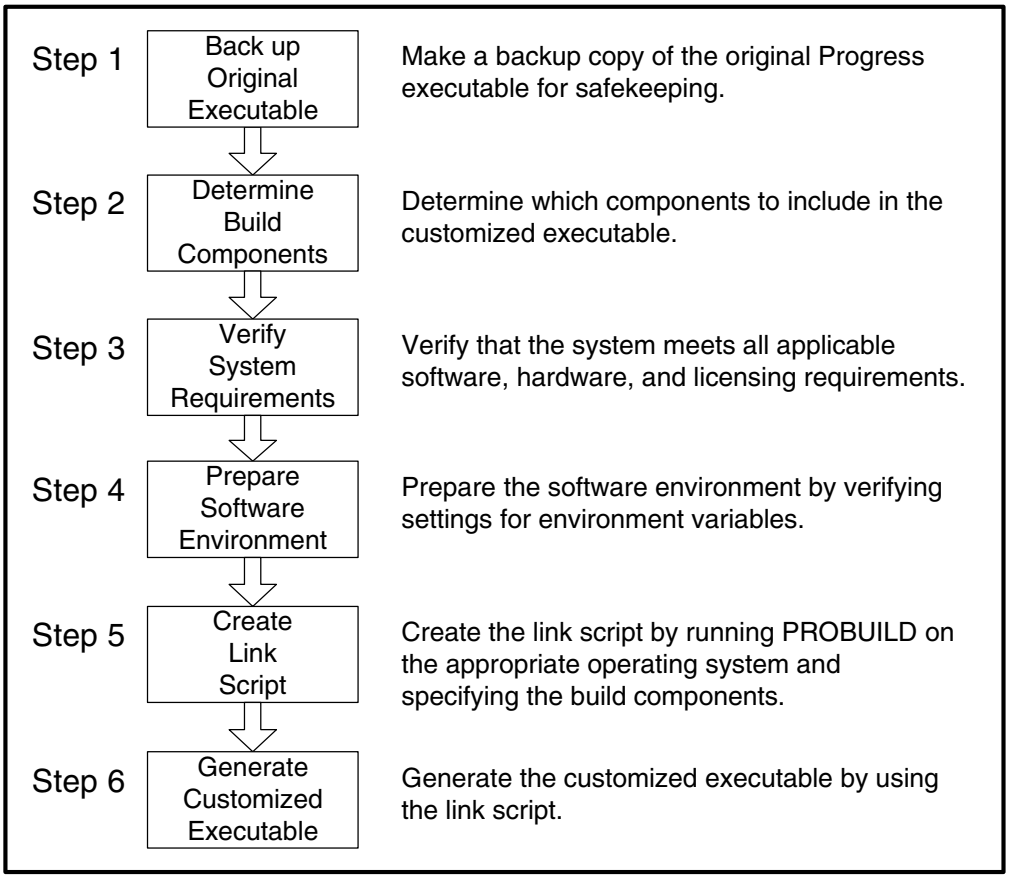



**Figure A–1: Build Process Steps**

Follow these basic steps to build a customized executable:

1 ♦ Make a backup copy of the original Progress executable for safekeeping, if necessary.

2 ♦ Determine which of the following types of components to include in the customized executable:

- A Progress product that you are licensed to build

- Optional Progress capabilities (configurable elements) that you are licensed to build into a Progress product
- Object files that you develop and compile separately

You must specify these components when you create the link script using PROBUILD in Step 5.

See the "Running PROBUILD on UNIX" section for the list of Progress products and configurable elements you can build on UNIX. See the "Running PROBUILD in Windows" section for the list of Progress products and configurable elements you can build in Windows.

3 ♦ Verify the system requirements. The system must meet all applicable software, hardware, and licensing requirements.

See the "Software and Hardware Requirements" section for information about system requirements. See the "Licensing Requirements" section for information about licensing.

4 ♦ Prepare the software environment by verifying settings for environment variables.

See the "Preparing the UNIX Software Environment" section for information about preparing the UNIX software environment. See the "Preparing the Windows Software Environment" section for information about preparing the Windows software environment.

5 ♦ Create the link script by running PROBUILD on the appropriate operating system and specifying the build components. You will use this link script to generate the customized executable in the next step.

See the "Running PROBUILD on UNIX" section for instructions on running PROBUILD on UNIX. See the "Running PROBUILD in Windows" section for instructions on running PROBUILD in Windows.

6 ♦ Generate the customized executable by using the link script on the appropriate operating system.

See the "Linking on UNIX" section for instructions on running the link script on UNIX. See the "Linking in Windows" section for instructions on running the link script in Windows.

For information about the linker utility, see the documentation supplied with the operating system.

## A.2 Software and Hardware Requirements

For Progress disk space and memory requirements, see the *Progress Installation and Configuration Guide Version 9 for UNIX* or the *Progress Installation and Configuration Guide Version 9 for Windows*. The memory requirements are the same as those for the Progress 4GL Client, plus any additional memory that the configurable elements require.

For example, a customized executable might require up to two times the size of the basic executable of free disk storage in addition to what the Progress product and other components require. Also, reserve sufficient disk space for object and temporary files created when you compile and link a component.

Table A–1 lists the software requirements for running PROBUILD.

**Table A–1: PROBUILD Software Requirements**

| Operating System | Software Component |
|---|---|
| UNIX | C compiler available with the operating system. (Required only for building HLC, ORACLE DataServer, and ESQL-89 modules.)<br>Linker available with the operating system.<br>C run-time libraries.<br>Communications libraries from the operating system vendor. |
| Windows | Microsoft Visual C++ Compiler Version 6.0 for Intel.<br>Linker included with the compiler.<br>Libraries included with the compiler. |

## A.3 Licensing Requirements

You are obligated to ensure that the customers who purchase your product have purchased and installed a properly licensed version of Progress before you install any Progress executables you deliver as part of your product.

## A.4 PROBUILD Overview

PROBUILD helps you to build customized versions of Progress executables by letting you select the executable you want to build and offering you a set of optional capabilities (or configurable elements) to include in that executable. PROBUILD then creates a link script. A *link script* is a file that specifies all the object modules that constitute an executable.

PROBUILD creates link scripts that are specific to the operating system where you run the utility. On UNIX, PROBUILD creates a shell script. In Windows, PROBUILD creates a linker response file (with a .lnk extension). Run PROBUILD on UNIX to create a link script for UNIX, and run PROBUILD in Windows to create a link script for Windows. You then use the link script to generate the actual executable.

## A.5 Building Executables on UNIX

This section provides instructions for building a customized executable on UNIX. It contains the following sections:

- Preparing the UNIX Software Environment
- Running PROBUILD on UNIX
- Linking on UNIX

### A.5.1 Preparing the UNIX Software Environment

When you install Progress, the installation program prepares the software environment. In the UNIX software environment, Progress uses the `buildenv` script to maintain environment variables.

The `buildenv` script sets default values for environment variables that define objects, libraries, and options required for building executables on UNIX. You can edit this script to customize environments at different sites. The `buildenv` script is located in the `$DLC/probuild/eucapp` directory.

Perform the following tasks to prepare the UNIX software environment:

- Verify that the `buildenv` script includes proper settings for the PROLOAD environment variable, which defines the PROBUILD installation directory (`$DLC/probuild`, by default), and any other customized environment variables. When run, the link script that PROBUILD creates on UNIX calls the `buildenv` script.
- Verify that the `probuild` script updates the PROPATH environment variable to include the `euc.pl` procedure library in the `$DLC/probuild/eucapp` directory. If PROPATH does not contain both the `eucapp` directory and the `euc.pl` procedure library, PROBUILD will not run.
- Verify that the PATH environment variable contains the path to the C++ compiler.

For more information about maintaining environment variables in the `buildenv` script, see Chapter 5, “Maintaining User Environments.”

For more information about UNIX environment variables, see the *Progress Installation and Configuration Guide Version 9 for UNIX*.

### A.5.2 Running PROBUILD on UNIX

Run PROBUILD on UNIX to create a link script from which you will generate a customized executable.

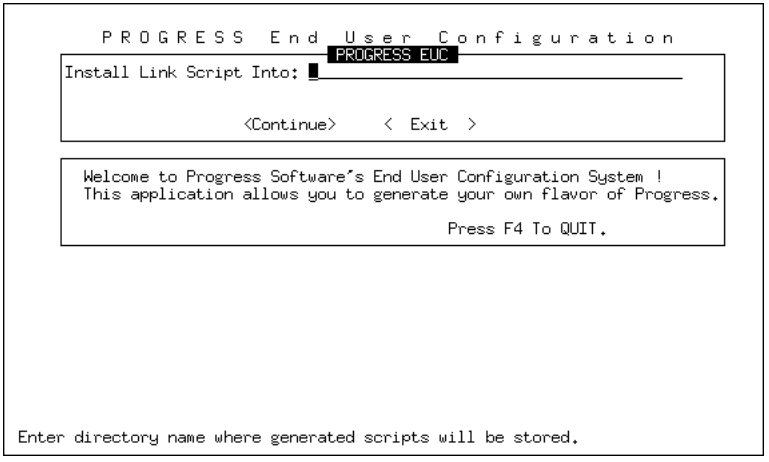
**NOTE:** Some executables you customize with PROBUILD require that you set the set-userid bit. To set the set-userid bit, log in as root or enter the root password when prompted.

To create a link script on UNIX:

1 ♦ Run PROBUILD by entering the following command on the command line:

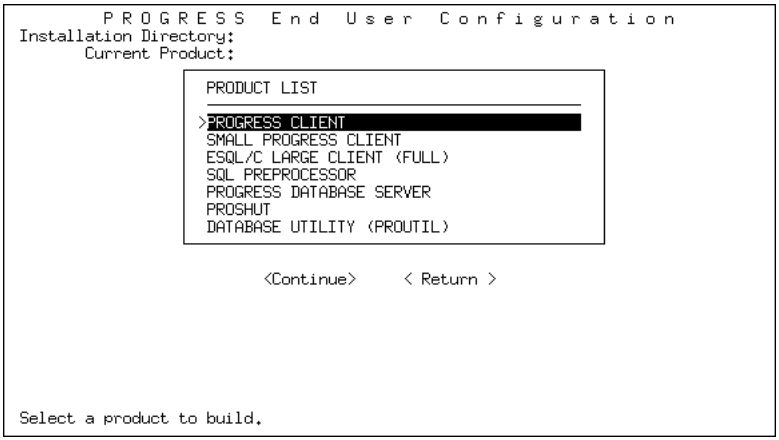
```
$PROLOAD/eucapp/probuild
```

The Progress EUC dialog box appears:




2 ♦ In the Install Link Script Into field, type the name of the directory in which PROBUILD will put the link script, and ultimately, the customized executable. Your current directory is the default. If you enter a directory that does not exist, the utility creates it for you.

3 ♦ Choose Continue. The Product List dialog box appears:




The product list contains the products that you are licensed to build. Table A–2 presents all the Progress products you can build on UNIX organized in order of appearance.

**Table A–2: PROBUILD Product List on UNIX** *(1 of 3)*

| Product | Description |
|---|---|
| Progress Client | The standard Progress client executable. The Progress Client includes the Progress 4GL Compiler.<br>Default link script name: `ldpro`<br>Default executable name: `_progres` |
| Small Progress Client | A run-time version of the standard Progress Client executable. The Small Progress Client does not include the Progress 4GL Compiler.<br>Default link script name: `ldprort`<br>Default executable name: `_progres` |
| ESQL/C Large Client (Full) | A full Progress Client executable that contains ESQL-89 application code and includes the SQL compiler. The ESQL/C Large Client processes static and dynamic SQL requests directly.<br>Default link script name: `ldesqlc`<br>Default executable name: `fullesql` |

**Table A–2: PROBUILD Product List on UNIX** *(2 of 3)*

| Product | Description |
|---|---|
| SQL Preprocessor | An executable that converts ESQL-89 source files to C source files that you can compile and link into an executable.<br>Default link script name: `ldsqlcpp`<br>Default executable name: `sqlcpp` |
| Progress Database Server | The standard Progress database server executable that provides a client with access to a Progress database.<br>Default link script name: `ldprosrv`<br>Default executable name: `_mprosrv` |
| PROSHUT | A utility executable that lets you stop server and broker processes. PROSHUT also lets you disconnect users before shutting down a database.<br>Default link script name: `ldprshut`<br>Default executable name: `_mprshut` |
| Database Utility (PROUTIL) | A utility executable that lets you perform various database administration operations on a Progress database, such as:<br>• Index maintenance<br>• Database conversion<br>• Database state and usage inquiry<br>Default link script name: `ldproutl`<br>Default executable name: `_proutil` |
| Remote DataServer Broker | A broker executable that provides a client with access to non-Progress database servers. The Remote DataServer Broker starts the appropriate type of database server for a client based on the type of client request.<br>Default link script name: `ldprobrkr`<br>Default executable name: `_probrkr` |
| Remote ORACLE DataServer | A database server executable that provides a client with access to an ORACLE database.<br>Default link script name: `ldorasrv`<br>Default executable name: `_orasrv` |

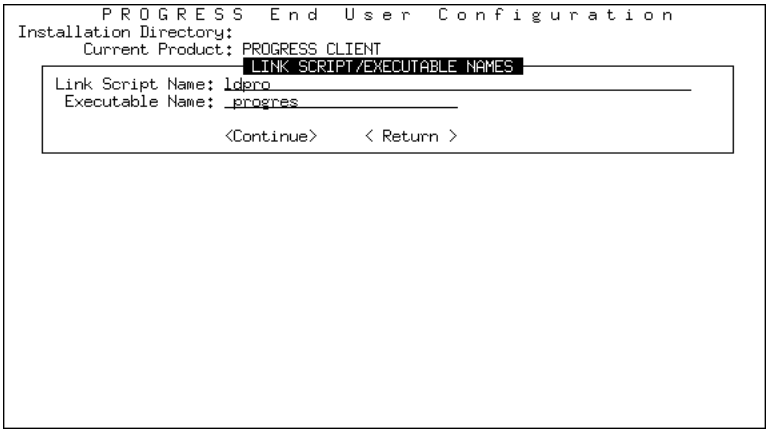
**Table A–2: PROBUILD Product List on UNIX** *(3 of 3)*

| Product | Description |
|---|---|
| Open Interface Driver | A server executable that processes static and dynamic SQL requests for an ESQL/C Small Client.<br>Default link script name: `ldoidrvr`<br>Default executable name: `_prooidv` |
| ESQL/C Small Client | A Progress Client executable that contains ESQL-89 application code and Progress client communications modules that provide access to the Progress Open Interface Driver (OID). The ESQL/C Small Client does not include the SQL compiler, it relies on the OID to process static and dynamic SQL requests.<br>Default link script name: `ldesqlnet`<br>Default executable name: `esqlnet` |
| Progress AppServer | The standard Progress application server executable. The Progress AppServer processes remote procedure requests for a client or another application server.<br>Default link script name: `ldapsv`<br>Default executable name: `_proapsv` |
| HLC/ESQL/C Client | A client executable that accesses Progress through an HLC or ESQL-89 interface.<br>Default link script name: `ldprol`<br>Default executable name: `_progrsl` |

4 ♦ Select the product you want to customize from the product list. Use the arrow keys to scroll through the list. Press **ENTER** to select the product. Press **TAB** to move to a button, then press **ENTER** to execute.

**NOTE:** You can create a link script for only one product at a time. After you create a link script for one product, you can create link scripts for additional products individually.

5 ♦ Choose Continue. The Link Script/Executable Names dialog box appears:



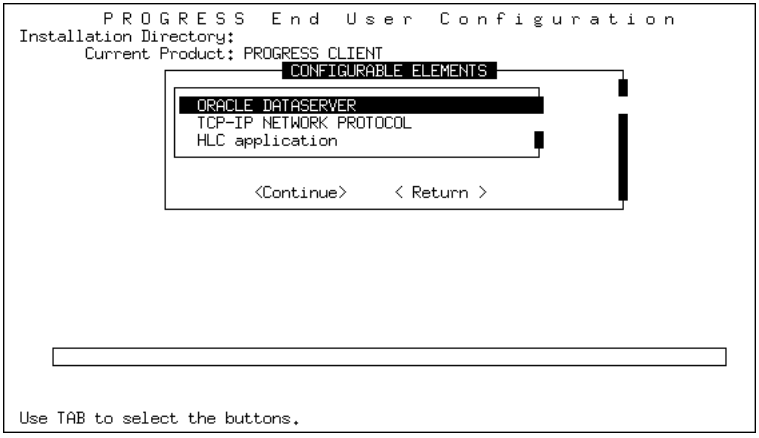

6 ♦ In the Link Script Name field, type a name for the link script or accept the default name.

The link script name is the name of the link script that PROBUILD creates and that you use to generate the customized executable. PROBUILD uses a different default link script name for each type of executable on each supported operating system. For example, `ldpro` is the default name for the Progress Client link script on UNIX systems.

7 ♦ In the Executable Name field, type a name for the executable or accept the default name.

The executable name is the name of the customized executable. PROBUILD does not generate this executable. PROBUILD inserts the name you specify here into the link script, which you run to generate the executable image. You can change the executable name later by editing the link script directly. PROBUILD uses a different default executable name for each type of executable on each supported operating system. For example, `_progres` is the default name for the Progress Client executable on UNIX systems.

8 ♦ Choose Continue. The Configurable Elements dialog box appears:




The configurable elements list contains the optional capabilities that you are licensed to build into the currently selected product. Each product has its own combination of appropriate configurable elements. Table A–3 presents all the Progress configurable elements you can build on UNIX organized in order of appearance.

**Table A–3: Configurable Elements on UNIX** *(1 of 2)*

| Configurable Element | Purpose |
|---|---|
| ORACLE DataServer | Include in the client executable to access a local ORACLE DataServer. |
| TCP/IP Network Protocol | Include in the executable to provide support for the TCP/IP network protocol. |
| HLC Application | Include in the client executable to provide support for calling C routines from within Progress 4GL code. |
| Remote ORACLE DataServer | Include in the client executable to access a remote ORACLE DataServer. |
| Named Pipes | Include in the executable if you want a Progress application to read from or write to a named pipe. |
| Remote DB2/400 DataServer | Include in the client executable to access a remote DB2/400 DataServer. |

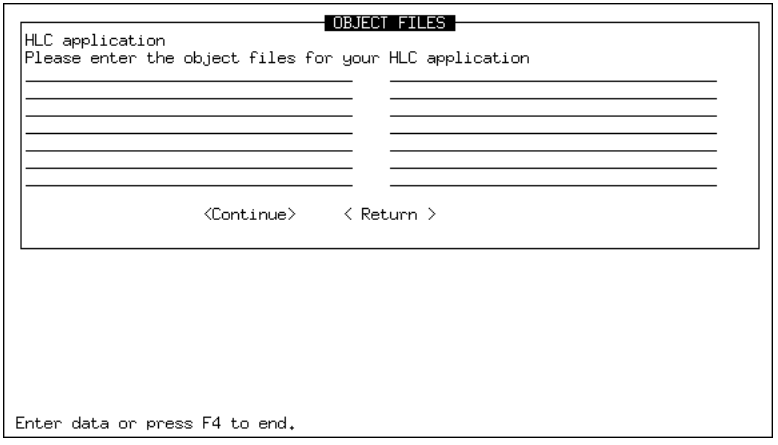
**Table A–3: Configurable Elements on UNIX** *(2 of 2)*

| Configurable Element | Purpose |
|---|---|
| User-defined Sort Routine | Include in the executable if the collation techniques provided by Progress do not meet your needs. |
| DataServer Library | Include in the client executable to access a local or remote ODBC DataServer. |
| No Debugger Comm Modules | Include in the executable if the executable will not support 4GL debugging capabilities. Debugging capabilities require networking software. This element excludes the dependence on networking. |

9 ♦ Select the configurable elements you want to build into the currently selected product. Use the **SPACEBAR** or the mouse to deselect an item.

10 ♦ Choose Continue.

If you selected a product or configurable element that requires one or more object files that you developed and compiled separately, PROBUILD prompts you to identify these object files in the Object Files dialog box:




Type in the object filenames, including the file extension required on the operating system, and choose Continue. PROBUILD includes these objects in the link script.

**NOTE:** If you are not prepared to list an object file, enter a comment line in the object field. PROBUILD includes the comment line in the link script. You can edit the link script later to replace the comment line with the actual object filename. Use comment characters if you want to run the link script before you replace the comment line. The link script might not run if you use only placeholder text for the comment line.

PROBUILD creates the specified link script. It also returns to the Product List dialog box where you can create another link script, or choose Return to return to the Progress EUC dialog box.

11 ♦ From the Progress EUC dialog box, you can enter another link script directory or choose Exit to exit PROBUILD.

### A.5.3 Linking on UNIX

Before you can use a customized executable, you must generate it using the link script you created with PROBUILD.

To generate the customized executable, use the linker utility to run the link script by entering the link script name on the command line. For example, generate the Progress Client executable using the following command:

```
ldpro
```

The linker utility generates the customized executable as specified in the link script. You can now use the customized executable.

For link utility error and troubleshooting information, see the documentation supplied with the operating system.

## A.6 Building Executables in Windows

This section provides instructions for building a customized executable in Windows. It contains the following sections:

- Preparing the Windows Software Environment
- Running PROBUILD in Windows
- Linking in Windows

### A.6.1 Preparing the Windows Software Environment

When you install Progress, the installation program prepares the software environment. In the Windows software environment, Progress uses the `progress.ini` file as well as the Registry to maintain environment variables.

The `progress.ini` file sets default values for environment variables that define objects, libraries, and options required for building executables in Windows. The link script that PROBUILD creates in Windows uses environment variables defined in the `progress.ini` file. You can edit the `progress.ini` file to customize environments at different sites. The `progress.ini` file is located in the `%DLC%\probuild\eucapp` directory, by default.

When you install Progress, the installation program automatically updates the registry with the information in the `progress.ini` file that is shipped to you. Progress Software recommends that you maintain environment variables in both the `progress.ini` file and the registry. If you modify environment variables in the `progress.ini` file, you must update the information in the registry.

Although there are several methods for adding and deleting information from the registry directly, Progress Software recommends that you do not use these direct update methods. Instead, maintain the registry information by editing the `progress.ini` file and using the INI2REG utility to update the registry. This approach synchronizes the information in the `progress.ini` file and the registry without causing unintended edits to the registry.

Perform the following tasks to prepare the Windows software environment:

- Verify settings for PROPATH and other environment variables in both the `progress.ini` file and the registry. For example, PROPATH must include the `euc.pl` procedure library in the `%DLC%\probuild\eucapp` directory. If PROPATH does not contain both the `eucapp` directory and the `euc.pl` procedure library, PROBUILD will not run.
- Verify that the Microsoft Visual C++ compiler is installed on the system.
- Verify that the PATH environment variable contains the path to the C++ compiler and the linker command (`link.exe`).

For more information about maintaining environment variables in the `progress.ini` file and the registry, and using the INI2REG utility, see Chapter 5, “Maintaining User Environments.”

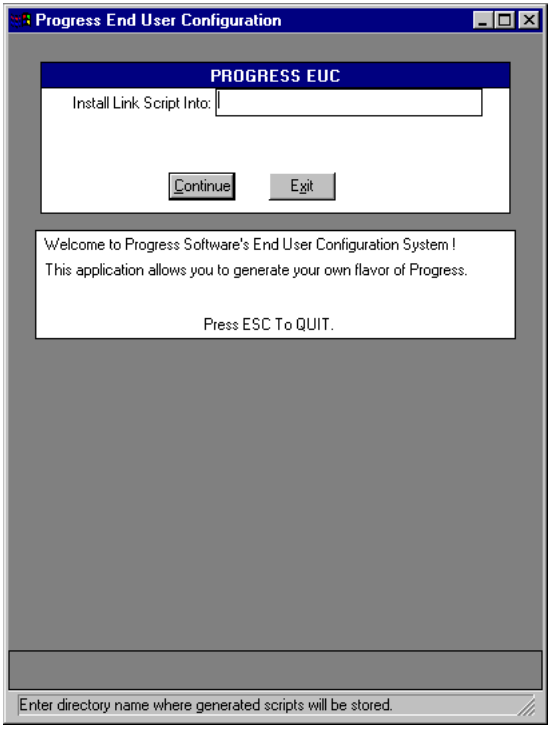
For information about Windows environment variables, see the *Progress Installation and Configuration Guide Version 9 for Windows*.

## A.6.2 Running PROBUILD in Windows

Run PROBUILD in Windows to create a link script from which you will generate a customized executable.

To create a link script in Windows:

1 ♦ Run PROBUILD by double-clicking the PROBUILD icon. The Progress EUC dialog box appears:

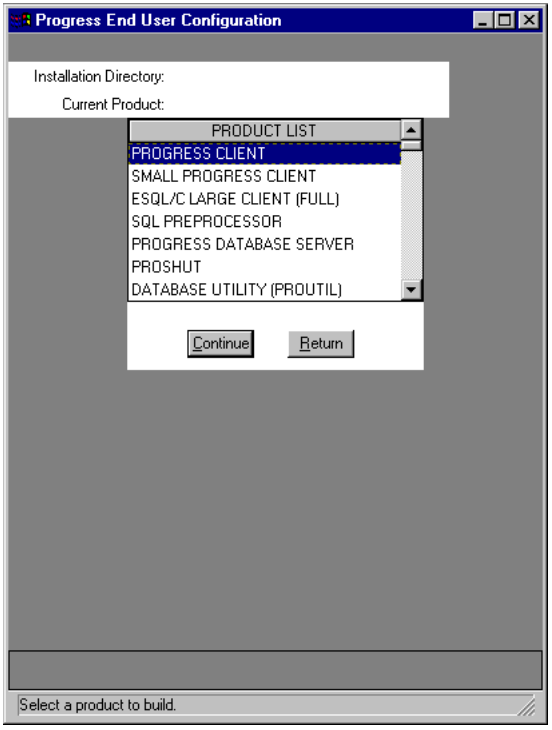



You can also run PROBUILD by entering the following command on the command line:

```
%DLC%\bin\prowin32.exe %DLC%\probuild\eucapp\euc.db -RO -p euc.p
```

2 ♦ In the Install Link Script Into field, type the name of the directory in which PROBUILD will put the link script, and ultimately, the customized executable. Your current directory is the default. If you enter a directory that does not exist, the utility creates it for you.

3 ♦ Choose Continue. The Product List dialog box appears:




The product list contains the Progress products you are licensed to build. Table A–4 presents all Progress products you can build in Windows organized in order of appearance.

**Table A–4: PROBUILD Product List in Windows** *(1 of 3)*

| Product | Description |
|---|---|
| Progress Client | The standard Progress Client executable. The Progress Client includes the Progress 4GL Compiler.<br><br>Default link script name: `prowin32.lnk`<br>Default executable name: `prow32.dll`<br><br>**NOTE:** The Progress Client link script generates a dynamic link library, not an executable. The Progress Client executable (`prowin32.exe`) calls `prow32.dll`.<br><br>To use the Progress Client executable with a customized version of `prow32.dll`:<br><br>• Make a backup copy of the original `prow32.dll`<br>• Replace the original version of `prow32.dll` with the customized version. Be sure to name the customized version `prow32.dll` and place it in the same directory as `prowin32.exe`.<br>• Run `prowin32.exe`. |
| Small Progress Client | A run-time version of the standard Progress Client executable. The Small Progress Client does not include the Progress 4GL Compiler.<br><br>Default link script name: `prort.lnk`<br>Default executable name: `prwinr32.exe` |
| ESQL/C Large Client (Full) | A full Progress Client executable that contains ESQL-89 application code and includes the SQL compiler.<br><br>The ESQL/C Large Client processes static and dynamic SQL requests directly.<br><br>Default link script name: `wcesql.lnk`<br>Default executable name: `esqltllg.exe` |
| SQL Preprocessor | An executable that converts ESQL-89 source files to C source files that you can compile and link into an executable.<br><br>Default link script name: `sqlcpp.lnk`<br>Default executable name: `sqlcpp.exe` |

**Table A–4: PROBUILD Product List in Windows** *(2 of 3)*

| Product | Description |
|---|---|
| Progress Database Server | The standard Progress database server executable that provides a client with access to a Progress database.<br>Default link script name: `mprosrv.lnk`<br>Default executable name: `_mprosrv.exe` |
| PROSHUT | A utility executable that lets you stop server and broker processes. PROSHUT also lets you disconnect users before shutting down a database.<br>Default link script name: `proshut.lnk`<br>Default executable name: `_mprshut.exe` |
| Database Utility (PROUTIL) | A utility executable that lets you perform various database administration operations on a Progress database, such as:<br>• Index maintenance<br>• Database conversion<br>• Database state and usage inquiry<br>Default link script name: `proutil.lnk`<br>Default executable name: `_proutil.exe` |
| Remote DataServer Broker | A broker executable that provides a client with access to non-Progress database servers. The Remote DataServer Broker starts the appropriate type of database server for a client based on the type of client request.<br>Default link script name: `probrkr.lnk`<br>Default executable name: `_probrkr.exe` |
| Remote ORACLE DataServer | A database server executable that provides a client with access to an ORACLE database.<br>Default link script name: `orasrv.lnk`<br>Default executable name: `_orasrv.exe` |
| Open Interface Driver | A server executable that processes static and dynamic SQL requests for an ESQL/C Small Client.<br>Default link script name: `oidrvr32.lnk`<br>Default executable name: `oidrvr32.exe` |

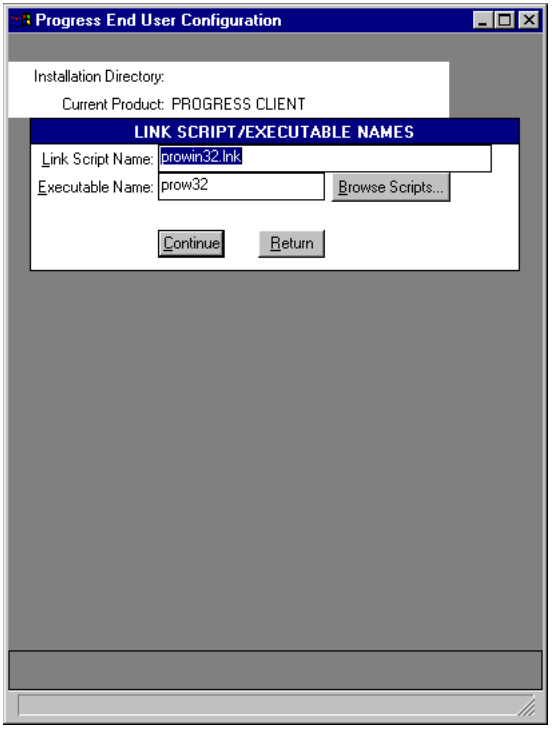
**Table A–4: PROBUILD Product List in Windows** *(3 of 3)*

| Product | Description |
|---|---|
| ESQL/C Small Client | A Progress Client executable that contains ESQL-89 application code and Progress client communications modules that provide access to the Progress Open Interface Driver (OID).<br><br>The ESQL/C Small Client does not include the SQL compiler, it relies on the OID to process static and dynamic SQL requests.<br><br>Default link script name: `esqlnet.lnk`<br>Default executable name: `esqlnet.exe` |
| Progress AppServer | The standard Progress application server executable. The Progress AppServer processes remote procedure requests for a client or another application server.<br><br>Default link script name: `apsv.lnk`<br>Default executable name: `_proapsv.exe` |
| Progress Character Mode Windows Client | The Progress character client executable. The Progress Character Mode Windows Client includes the Progress 4GL Compiler.<br><br>Default link script name: `prchar.lnk`<br>Default executable name: `_prchar.exe` |
| HLC/ESQL/C Client | A client executable that accesses Progress through an HLC or ESQL-89 interface.<br><br>Default link script name: `prown132.lnk`<br>Default executable name: `prown132.exe` |

4 ♦ Select the product you want to customize from the product list. Use the arrow keys to scroll through the list. Press **ENTER** to select the product. Press **TAB** to move to a button, then press **ENTER** to execute.

**NOTE:** You can create a link script for only one product at a time. After you create a link script for one product, you can create link scripts for additional products individually.

5 ♦ Choose Continue. The Link Script/Executable Names dialog box appears:




6 ♦ In the Link Script Name field, type a name for the link script or accept the default name. Choose the Browse Scripts button to view your directories and files before choosing a link script name.

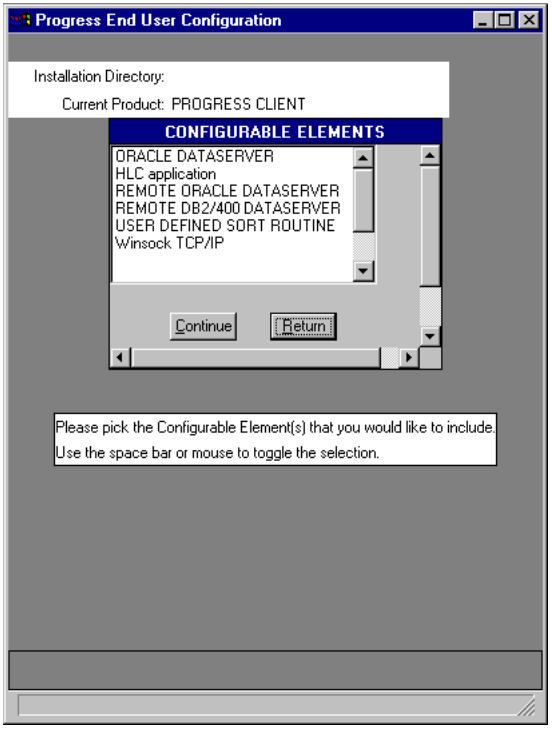
The link script name is the name of the link script that PROBUILD creates and that you use to build the customized executable. PROBUILD uses a different default link script name for each type of executable on each supported operating system. For example, `prchar.lnk` is the default name for the Progress Character Mode Windows Client link script.

7 ♦ In the Executable Name field, type a name for the executable or accept the default name.

The executable name is the name of the customized executable. PROBUILD does not generate this executable. PROBUILD inserts the name you specify here into the link script, which you run to generate the executable image. You can change the executable name later by editing the link script directly. PROBUILD uses a different default executable name for each type of executable on each supported operating system. For

example, `_prchar.exe` is the default name for the Progress Character Mode Windows Client executable.

8 ♦ Choose Continue. The Configurable Elements dialog box appears:




The configurable elements list contains the optional capabilities that you are licensed to build into the currently selected product. Each product has its own combination of appropriate configurable elements. Table A–5 describes all the Progress configurable elements you can build in Windows.

**NOTE:** The exact set of configurable elements that appear in the list depends on the Progress products you have purchased.

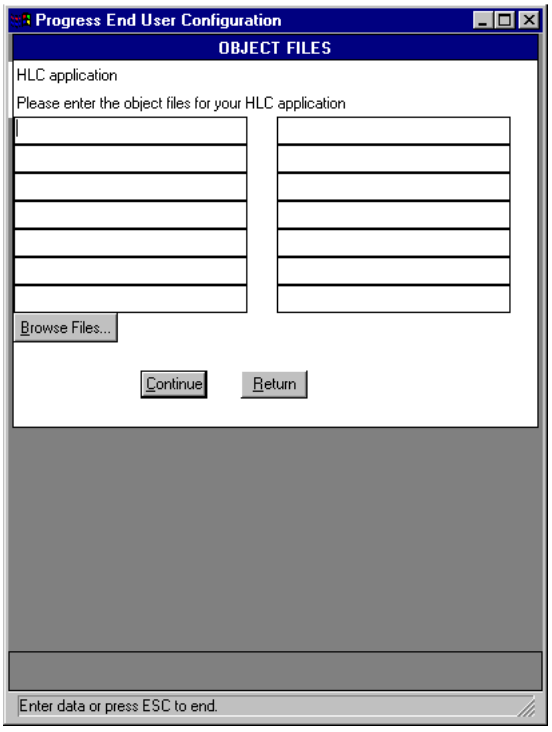
**Table A–5: Configurable Elements in Windows**

| Configurable Element | Purpose |
|---|---|
| ORACLE DataServer | Include in the client executable to access a local ORACLE DataServer. |
| CTOS Network Protocol | Include in the executable to provide support for the CTOS network protocol. |
| HLC Application | Include in the client executable to provide support for calling C routines from within Progress 4GL code. |
| Remote ORACLE DataServer | Include in the client executable to access a remote ORACLE DataServer. |
| Remote DB2/400 DataServer | Include in the client executable to access a remote DB2/400 DataServer. |
| User-defined Sort Routine | Include in the executable if the collation techniques provided by Progress do not meet your needs. |
| Winsock TCP/IP | Include in the executable to provide support for the Winsock TCP/IP network protocol. |
| DataServer Library | Include in the client executable to access a local or remote ODBC DataServer. |
| ODBC DataServer | Include in the client executable to access a local ODBC DataServer. |
| No Debugger Comm Modules | Include in the executable if the executable will not support 4GL debugging capabilities. Debugging capabilities require networking software. This element excludes the dependence on networking. |

9 ♦ Select the configurable elements you want to build into the currently selected product. Use the **SPACEBAR** or the mouse to deselect an item.

10 ♦ Choose Continue.

If you selected a product or configurable element that requires one or more object files that you developed and compiled separately, PROBUILD prompts you to identify these object files in the Object Files dialog box:




Type in or browse for the object filenames, including the file extension required on the operating system, and choose Continue. PROBUILD includes these objects in the link script.

**NOTE:** If you are not prepared to list an object file, enter a comment line in the object field. PROBUILD includes the comment line in the link script. You can edit the link script later to replace the comment line with the actual object filename. Use comment characters if you want to run the link script before you replace the comment line. The link script might not run if you use only placeholder text for the comment line.

PROBUILD creates the specified link script. It also returns to the Product List dialog box where you can create another link script, or choose Return to return to the Progress EUC dialog box.

11 ♦ From the Progress EUC dialog box, you can enter another link script directory or choose Exit to exit PROBUILD.

### A.6.3 Linking in Windows

Before you can use a customized executable, you must generate it using the link script you created with PROBUILD.

To generate the customized executable, use the linker utility to run the link script using the following syntax:

```
link @link-script-filename
```

For example, the following command generates the Progress Character Mode Windows Client executable:

```
link @prchar.lnk
```

The linker utility generates the customized executable as specified in the link script. You can now use the customized executable.

For link utility error and troubleshooting information, see the documentation supplied with the operating system.

## A.7 Troubleshooting

Following are some common errors that might occur when you attempt to link an executable:

- An object or library file cannot be found.
- An environment variable is not properly set.
- The system has insufficient memory or disk space.

For link utility error and troubleshooting information, see the documentation supplied with the operating system.

# B

# Progress Application Limits

The hardware and operating system version on which applications run often determines the limits for Progress applications. This appendix lists the Progress limits you must consider when developing a Progress application. The limits are organized in the following categories:


- "Input/Output Limits"
- "Sorting Limits"
- "Name Limits"
- "Compiler Limits"


For information about Progress database limits, see the *Progress Database Administration Guide and Reference*.

## B.1 Input/Output Limits

Table B–1 lists the maximum number of characters you can use for input and output devices (unless otherwise specified).

**Table B–1: Input/Output Limits**

| Device | Operating System | Limit |
|---|---|---|
| Terminal line width | UNIX | 80 to 132 columns |
| | Windows | 80 columns |
| Printer line width | All | 1 to 255 columns |
| Export file | All | 1 to 3,000 characters per field; 32K bytes per record |
| Import file | All | 1 to 3,000 characters per field; 32K bytes per record |
| Streams | All | 1 to 5 streams per procedure |

**NOTE:** Windows NT batch files limit you to no more than nine (9) input parameters. This is a Windows limitation. Actual Progress executables handle more than nine.

## B.2 Sorting Limits

Table B–2 lists the limits for sorting data (unless otherwise specified).

**Table B–2: Sorting Limits**

| Category | Limit |
|---|---|
| Levels | 1 to 16 columns or expressions. |
| Size | Typically, 1 to 197 bytes (that is, approximately 200 bytes minus some overhead per column and expression). The size limit depends on the number of columns and expressions you are sorting. Each column and expression uses up some number of available bytes. |

## B.3 Name Limits

Table B–3 lists the maximum number of characters you can use in names (unless otherwise specified).

**Table B–3: Name Limits**

| Name Type | Operating System | Limit |
|---|---|---|
| External procedure | UNIX | 1 to 58 characters, plus a 2 character extension. |
| | Windows | 1 to 255 characters, plus a 2 character extension. |
| Pathnames | All | 1 to 255 characters. |
| Table<br>Field<br>Field-level widget<br>Index<br>Variable<br>Run-time procedure parameter | All | 1 to 32 characters; can consist of any combination of letters (a-z or A-Z), numbers (0-9), and these special characters: # $ - _ % &<br>Names must begin with a letter. |
| Internal procedure<br>Block label<br>Query<br>Frame widget<br>Browse widget<br>Menu widget<br>Submenu widget<br>Menu item widget | All | No set limit; can consist of any combination of letters (a-z or A-Z), numbers (0-9), and these special characters: # $ - _ % &<br>Names must begin with a letter. |
| Stream | All | 1 to 12 characters; can consist of any combination of letters (a-z or A-Z), numbers (0-9), and these special characters: # $ - _ % &<br>Names must begin with a letter. |
| Tables in the convmap file:<br><br>Attribute table<br>Collation table<br>Conversion table<br>Case table | All | 1 to 20 characters; can consist of any combination of letters (a-z or A-Z), numbers (0-9), and the hyphen character (-).<br>Names must begin with a letter. |

## B.4 Compiler Limits

Table B–4 lists the limits for each element of the Compiler (unless otherwise specified).

**Table B–4: Compiler Limits**

| Element | Limit |
|---|---|
| Variables | 32,000 bytes for UNDO and NO-UNDO variables per external procedure. 32,000 bytes for local UNDO and local NO-UNDO variables per internal procedure or trigger block. |
| Statements | 1 to 32,000 characters per statement. Use the Input Characters (-inp) startup parameter to limit the number of characters allowed in a single statement.<br>The number of tokens allowed per statement is limited only by the available system resources. Each word or special character, such as a parenthesis, plus sign, and minus sign, counts as one token. Use the Token (-tok) startup parameter to limit the number of tokens allowed in a single statement. |
| Frames | 1 to 320 characters. |
| Nested blocks | 1 to 255 blocks, including called procedures. Use the Nested Blocks (-nb) startup parameter to limit the number of nested blocks allowed in a procedure. |

For more information about the Input Characters (-inp), Token (-tok), and Nested Blocks (-nb) startup parameters, see the *Progress Startup Command and Parameter Reference*.

# Index

## A


Application security. *See* Security

Applications
  *See also* Building executables
  accessing databases 2–1
  administration 1–2
  deployment 1–2
  limits B–1
  maintaining user environments
    in Windows 5–2
    on UNIX 5–34
  performance 4–1

Audience ix

Auto-connect feature 2–11


## B


Blank user IDs 3–6

Bold typeface
  as typographical convention x

buildenv scripts
  and the PROBUILD utility A–5
  maintaining 5–35
  sample 5–35

Building executables
  *See also* Linking executables, PROBUILD utility
  build process A–2
  buildable products
    in Windows A–16
    on UNIX A–7
  building in Windows A–13
  building on UNIX A–5
  configurable elements
    in Windows A–21
    on UNIX A–11
  default names
    in Windows A–16
    on UNIX A–7
  hardware requirements A–4
  licensing requirements A–4
  software requirements A–4


## C


client.mon files 4–6

Colors. *See* progress.ini file, PROTERMCAP file

Compiler limits B–4

Compile-time security
  defined 3–1
  establishing 3–2

CONNECT statement 2–6
handling connection denials 2–13
precedence over auto-connect 2–12

Connecting
at startup 2–5
connection modes 2–3
connection parameters 2–4
connection techniques 2–5
from the Data Administration tool 2–9
from the Data Dictionary 2–7
handling connection denials 2–13
non-Progress databases 2–12
over wide area networks 2–14
reducing connection times 2–14
specifying logical database names 2–2
with auto-connect 2–11
with the CONNECT statement 2–6

Connection parameters. *See* Startup parameters

Connection security, defined 3–1

CRC. *See* Cyclic redundancy check

Cyclic redundancy check (CRC)
and r-code 3–2

## D

Data Administration tool
connecting databases 2–9
defining compile-time security 3–2
determining privileges 3–6
disconnecting databases 2–17
selecting a working database 2–16

Data Dictionary
connecting databases 2–7
defining compile-time security 3–2
determining privileges 3–6
disconnecting databases 2–16
performing security administration 3–8
selecting a working database 2–15
temporary changes (.trp) file 4–11

Databases
connecting 2–2
disconnecting 2–16
file security 3–1
logical names 2–2
non-Progress 2–12
schemas 2–14
selecting a working database 2–15, 2–16
startup without connecting 2–15
temporary files 4–11

.dbi files 4–11

Default window characteristics. *See* progress.ini file

Defining debugger buttons. *See* progress.ini file

Defining debugger macros. *See* progress.ini file

Deployment
*See also* Applications
process 1–2
requirements 1–2
services 1–2
tasks 1–2

Direct printing, defined 7–2

Disconnecting databases 2–16
with the Data Administration Tool 2–17
with the Data Dictionary 2–16

## E

Edit buffer contents files 4–11

Environment characteristics. *See* progress.ini file

Environment variables, specifying 5–32

Error messages
displaying descriptions xv

Executables. *See* Building executables, Linking executables, PROBUILD utility

Extended character support
with progress.ini file entries 5–30
with PROTERMCAP file entries 5–61

## F

File descriptor limits 2–2

File system recovery program (fsck) 4–11

Files
buildenv scripts A–5
client.mon 4–6
distributing files on a network 4–9
edit buffer contents files 4–11
link scripts A–5
linker response files A–5
local before-image files 4–11
parameter files 2–4
printcap files 7–5
printer control files 7–12
proc.mon 4–6
procedure library (.pl) files 6–5
progress.ini file 4–10, 5–8
PROTERMCAP file 5–36
sort (.srt) files 4–3, 4–11
temporary changes files 4–11
temporary files 4–11
temporary table files 4–11

Fonts. *See* progress.ini file

fsck (file system recovery program). *See* File system recovery program (fsck)

## H

Help, Progress messages xv

## I

INI2REG Utility
*See also* Registry
command line interface 5–4
graphical interface 5–2

Input/output device limits
number of characters B–2

Italic typeface
as typographical convention x

## K

Key bindings. *See* progress.ini file, PROTERMCAP file

Keymapping
*See also* progress.ini file, PROTERMCAP file
Version 6 5–50

Keystrokes xi

## L

.lbi files 4–11

Libraries
*See also* PROLIB utility, R-code
advantages 4–2
and code pages 6–21
and PROPATH 6–8
and r-code performance 4–2
listing contents 6–16
memory and network considerations 6–9
memory-mapped format 6–2
advantages 6–2
generating 6–13
loading and executing 4–4, 6–4
overview 6–2
running procedures 6–6

Libraries (continued)
standard format 6–2
adding files 6–14
advantages 6–2
compressing files 6–19
creating 6–12
deleting files 6–15
extracting files 6–17
loading and executing 4–3, 6–3
replacing files 6–15
tuning 4–5

LIBRARY function 6–7

Limits
compiler B–4
input/output devices
number of characters B–2
name limits
number of characters B–3
sorting B–2

Link script, defined A–5

Linker response file, defined A–5

Linking executables
*See also* Building executables, PROBUILD utility
creating link scripts
in Windows A–13
on UNIX A–5
creating linker response files A–13
linking in Windows A–24
linking on UNIX A–13
troubleshooting A–24

Local before-image files 4–11, 4–12

Logical database names 2–2
changing with the ALIAS statement 2–3

LP Scheduler 7–5

lp spooler, defined 7–5

lpd daemons, defined 7–5

lpr spooler, defined 7–5

## M

Manual
organization of ix
syntax notation xi

MEMBER function 6–8

Memory-mapped library. *See* Libraries

Messages, displaying descriptions xv

Monospaced typeface
as typographical convention xi

## N

Name limits
number of characters B–3

Non-English character support 5–61

Non-Progress databases 2–12

## O

Operating system security 3–7

Output routing 7–8

OUTPUT THROUGH statement 7–4, 7–6

OUTPUT TO statement 7–3, 7–6
PRINTER option 7–3

## P

Parameter files. *See* Startup parameters

Performance
distributing files on a network 4–9
monitoring r-code activity 4–6
procedure loading and execution 4–2
reducing database connection times 2–14
sorting 4–12
temporary file I/O 4–11
using r-code libraries 4–2

Permissions
*See also* Security
blank user ID 3–6
CAN-CREATE 3–4
CAN-DELETE 3–4
CAN-DUMP 3–4
CAN-LOAD 3–4
CAN-READ 3–4
CAN-WRITE 3–4
table 3–7

.pge files 4–11

Print queues 7–2

printcap files 7–5

Printers
administration 7–1
configuration 7–12
control files 7–12
control sequences 7–12
defaults 7–3, 7–4

Printing 7–1
and UNIX networks 7–6
direct 7–2
in Progress 7–2
in Windows 7–8
on UNIX 7–5
output routing schemes 7–8
OUTPUT THROUGH statement 7–4, 7–6
OUTPUT TO statement 7–3, 7–6
pagination 7–4
PUT CONTROL statement 7–12
spooled 7–2
uux command 7–7

PROBUILD utility
*See also* Linking executables
build process A–2
buildable products
in Windows A–16
on UNIX A–7
building executables A–2
in Windows A–13
on UNIX A–5
configurable elements
in Windows A–21
on UNIX A–11
creating link scripts A–5
creating linker response files A–13
overview A–4
running
in Windows A–15
on UNIX A–6
startup commands
in Windows A–15
on UNIX A–6

proc.mon files 4–6

Procedure files, distributing 4–9

Procedure library (.pl) files. *See* Libraries, R-code

Procedures
distributing source files 4–9
running from a library 6–6

Progress system files distributing 4–9

progress.ini file
defining debugger buttons 5–28
defining debugger macros 5–28
defining key bindings 5–26
defining key mappings 5–30
maintaining 5–8
search path 5–7

progress.ini file (continued)
sections
colors section 5–19
debug-buttons section 5–28
debug-init section 5–28
debug-macros section 5–28
default window section 5–21
fonts section 5–21
keymap section 5–30
keys section 5–28
startup section 5–9
winchar colors section 5–25
winchar default window section 5–25
winchar keys section 5–26
winchar startup section 5–22
specifying colors 5–19, 5–25
specifying default window characteristics 5–21, 5–25
specifying environment characteristics 5–9, 5–22
specifying environment variables 5–32
specifying fonts 5–21
specifying key bindings 5–28

PROLIB utility 6–1
*See also* Libraries, R-code
parameters 6–10
sample commands 6–20
using 6–9
using wild cards 6–11

PROTERMCAP file
copying a UNIX termcap file 5–62
copying an existing terminal entry 5–64
defined 5–36
general syntax 5–38
keyboard mappings 5–53
maintaining 5–36
non-English character support 5–61
PROTERMCAP environment variable 5–64
setting the terminal type 5–63
size restrictions 5–37
specifying colors 5–50
syntax 5–37
terminal capabilities entries 5–41
for cursor motion 5–51
terminal name entries 5–40
Vermont Views 5–47

PROUTIL utility 3–2

PUT CONTROL statement 7–12

## R

R-code
*See also* Libraries
and code pages 6–21
and cyclic redundancy check (CRC) 3–2
defined 4–2
directory
defined 4–3
size 4–4
execution buffer, defined 4–3
execution environment 4–3
loading and executing 4–3, 6–2
from a file 4–3, 6–3
from a memory-mapped library 4–4, 6–4
from a standard library 4–3, 6–3
monitoring activity 4–6
performance 4–8
segment descriptor table, defined 4–3
session sort file, defined 4–3
shared memory buffer, defined 4–3
tuning 4–4

Registry
*See also* INI2REG Utility
updating 5–2

Remote printing
setting up 7–7

Run-time security
defined 3–1
establishing permissions tables 3–7

## S

Schema cache files 2–14

Schema holder, defined 2–12

Schema security, defined 3–1

SEARCH function 6–7

Security
*See also* Permissions
administration 3–8
application 3–1
blank user ID 3–6
compile-time 3–2
connection 3–1
database file 3–1
field-level 3–2
operating system 3–7
run-time 3–7
schema 3–1
table-level 3–2

Session files, distributing 4–10

Sort (.srt) files 4–3, 4–11

Sorting
improving performance 4–12
limits B–2

Source files, distributing 4–9

Spooled printing, defined 7–2

Spoolers
lp 7–5
lpd daemon 7–5
lpr 7–5

.srt files 4–11

Standard library. *See* Libraries

Startup parameters
Database Name (-db) 2–4
Directory Size (-D) 4–4
Host Name (-H) 2–3
Input Characters (-inp) B–4
Internal Code Page (-cpinternal) 6–12, 6–21
Logical Database Name (-ld) 2–2
Maximum Memory (-mmax) 4–4, 4–8
Merge Number (-TM) 4–12
Nested Blocks (-nb) 4–5, B–4
Number of Users (-n) 2–13
Parameter File (-pf) 2–4
Password (-P) 3–6
Printer (-o) 7–3, 7–4, 7–9

Startup parameters (continued)
PROLIB Memory (-plm) 4–5, 6–9
PROLIB Swap (-pls) 4–4, 4–5, 6–4, 6–9
Quick Request (-q) 6–8
R-code In Code Page (-cprcodein) 6–21
Save Temp Files (-t) 4–11
Schema Cache File (-cache) 2–14
Segment Statistics (-yd) 4–6, 4–8
Service Name (-S) 2–3
Single-user Mode (-1) 2–4
Speed Sort (-TB) 4–12
Stack Size (-s) 4–5
Statistics (-y) 4–6, 4–7
Statistics with Cross-reference (-yx) 4–6
Statistics with CTRL-C (-yc) 4–6
Temporary Directory (-T) 4–10, 4–11
Token (-tok) B–4
User ID (-U) 3–6
Windows Passthrough Printing (-wpp) 7–8

Syntax notation xi

## T

Table permissions
CAN-CREATE 3–4
CAN-DELETE 3–4
CAN-DUMP 3–4
CAN-LOAD 3–4
CAN-READ 3–4
CAN-WRITE 3–4

Temporary changes files 4–11

Temporary files
defined 4–11
distributing 4–10

Temporary table files 4–11

TERMINAL function 5–63

Terminals
*See also* PROTERMCAP file
UNIX stty control functions 5–60

.trp files 4–11

Typographical conventions x

## U

User count, exceeding 2–13

User environments
maintaining in Windows 5–2
maintaining on UNIX 5–34

Utilities
*See also* Data Administration tool, Data Dictionary
INI2REG 5–2
PROBUILD A–4

Utilities (continued)
PROLIB 6–1, 6–9
PROUTIL 3–2

uux command 7–7

## W

Wide area networks, connecting 2–14

Wild cards
and the PROLIB utility 6–11

Windows icons 5–33

Working databases, selecting 2–15