

Progress Developer's Toolkit

© 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: 4513
Item Number: 81081;9.1C

Contents

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

1. Introduction **1–1**
1.1 Toolkit Components 1–2

2. Choosing a Code Format 2–1
2.1 Unencrypted Source 2–2
2.1.1 Development Product Requirements 2–2
2.1.2 User Product Requirements 2–2
2.1.3 Advantages of Unencrypted Source 2–2
2.1.4 Possible Drawbacks of Unencrypted Source 2–3
2.2 Encrypted Source 2–4
2.2.1 Development Product Requirements 2–4
2.2.2 User Product Requirements 2–4
2.2.3 Advantages of Encrypted Source 2–5
2.2.4 Possible Drawbacks of Encrypted Source 2–6
2.3 R-code 2–6
2.3.1 Developer Product Requirements and R-code Portability 2–7
2.3.2 User Product Requirements 2–7
2.3.3 Advantages of R-code. 2–7
2.3.4 Possible Drawbacks of R-code 2–9

3. Initial Deployment 3–1
3.1 Unencrypted Source 3–3
3.2 Encrypted Source 3–5
3.3 CRC R-code 3–9
3.4 Time-stamp R-code 3–13

4. Upgrades 4–1
4.1 Modifying the Application Procedures 4–4
4.1.1 Unencrypted Source 4–4
4.1.2 Encrypted Source Files. 4–4
4.1.3 Time-stamp-based R-code 4–4
4.1.4 CRC-based R-code. 4–4
4.2 Modifying the Application Database 4–5
4.2.1 Upgrading with Updated .df and Encrypted Source 4–5
4.2.2 Upgrading with Updated .df and CRC-based R-code 4–8
4.2.3 Upgrading with Time-stamp-based R-code Files 4–10
4.2.4 Progress Product Upgrades 4–11

5. Deployment Topics and Tasks 5–1
5.1 Encrypting Procedures 5–2
5.1.1 XCODE Utility 5–2
5.1.2 Preparing to Use XCODE 5–2
5.1.3 Decryption and Compilation 5–3
5.2 BUNDLE Utility 5–4
5.2.1 The bundle.c Program 5–4
5.2.2 The unbundle.c Program 5–6
5.2.3 Examples Using BUNDLE 5–7
5.3 Dumping and Loading 5–11
5.3.1 MKDUMP Utility 5–12
5.3.2 Using Your Dump/Reload Facility 5–13
5.4 Restricting Database Access 5–14

A. Toolkit Contents and Reference A–1
A.1 Toolkit Templates A–2
A.2 Toolkit Utilities Summary A–3

Index Index–1

Figures

Figure 3–1: Deploying Database Structures and Application Procedures 3–2
Figure 4–1: Deploying Upgraded Database Structure and Application Procedures 4–3
Figure 5–1: Output of mkdump 5–12

Tables

Table A–1: Toolkit Utilities A–3

Preface

Purpose

This book describes the steps required to deploy Progress applications and the tradeoffs between different deployment scenarios. It also describes components provided with the Developer's Toolkit that help you through the deployment process.

Audience

This book is intended for experienced Progress programmers preparing applications for wider distribution or resale.

Organization of This Manual

Chapter 1, "Introduction"

Provides an introduction to the Developer's Toolkit.

Chapter 2, "Choosing a Code Format"

Describes the different types of code formats you can choose to deploy a Progress application.

Chapter 3, "Initial Deployment"

Describes the initial deployment process using the different code formats.

Chapter 4, "Upgrades"

Describes how to upgrade Progress applications.

Chapter 5, "Deployment Topics and Tasks"

Describes how to perform several tasks related to deploying Progress applications.

Appendix A, "Toolkit Contents and Reference"

Provides reference information about utilities and scripts provided by the Developer's Toolkit.

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 ... ] }
       }
```

Example Procedures

This manual provides numerous example procedures that illustrate syntax and concepts. Examples use the following conventions:

- They appear in boxes with borders.
- If they are available online, the name of the procedure appears above the left corner of the box and starts with a prefix associated with the manual that references it, as follows:
  - `e-` — *Progress External Program Interfaces*, for example, `e-ddeex1.p`
  - `lt-` — *Progress Language Tutorial*, for example, `lt-05-s3.p`
  - `p-` — *Progress Programming Handbook*, for example, `p-br01.p`
  - `r-` — *Progress Language Reference*, for example, `r-dynbut.p`
- If the name does not start with a listed prefix, the procedure is not available online.
- If they are not available online, they compile as shown, but might not execute for lack of completeness.

Accessing Files in Procedure Libraries on Windows Platforms

Documentation examples are stored in procedure libraries, `prodoc.pl` and `prohelp.pl`, in the `src` directory where Progress is installed.

You must first create all subdirectories required by a library before attempting to extract files from the library. You can see what directories and subdirectories a library needs by using the PROLIB -list command to view the contents of the library. See the *Progress Client Deployment Guide* for more details on the PROLIB utility.

Extracting source files from a procedure library involves running PROENV to set up your Progress environment, creating the directory structure for the files you want to extract, and running PROLIB.

1 ♦ From the Control Panel or the Progress Program Group, double-click the Proenv icon.

2 ♦ The Proenv Window appears, with the proenv prompt.

Running Proenv sets the DLC environment variable to the directory where you installed Progress (by default, `C:\Program Files\Progress`). Proenv also adds the DLC environment variable to your PATH environment variable and adds the bin directory `(PATH=%DLC%;%DLC%\bin;%PATH%)`.

3 ♦ Enter the following command at the proenv prompt to create the prodoc directory in your Progress working directory (by default, C:\Progress\Wrk):

```
MKDIR prodoc
```

4 ♦ Create the langref directory under prodoc:

```
MKDIR prodoc\langref
```

5 ♦ To extract all examples in a procedure library directory, run the PROLIB utility. Note that you must use double quotes because "Program Files" contains an embedded space:

```
PROLIB "%DLC%\src\prodoc.pl" -extract prodoc\langref\*.*
```

PROLIB extracts all examples into prodoc\langref.

To extract one example, run PROLIB and specify the file that you want to extract as it is stored in the procedure library:

```
PROLIB "%DLC%\src\prodoc.pl" -extract prodoc/langref/r-syshlp.p
```

PROLIB extracts r-syshlp.p into prodoc\langref.

Extracting Source Files from Procedure Libraries on UNIX Platforms

To extract p-wrk1.p from prodoc.pl, a procedure library, follow these steps at the UNIX system prompt:

1 ♦ Run the PROENV utility:

```
install-dir/dlc/bin/proenv
```

Running proenv sets the DLC environment variable to the directory where you installed Progress (by default, /usr/dlc). The proenv utility also adds the DLC environment variable to your PATH environment variable and adds the bin directory (PATH=%DLC%;%DLC%\bin;%PATH%).

2 ♦ At the proenv prompt, create the prodoc directory in your Progress working directory:

```
mkdir prodoc
```

3 ♦ Create the proghand directory under prodoc:

```
mkdir prodoc/proghand
```

4 ♦ To extract all examples in a procedure library directory, run the PROLIB utility:

```
prolib $DLC/src/prodoc.pl -extract prodoc/proghand/*.*
```

PROLIB extracts all examples into prodoc\langref.

To extract one example, run PROLIB and specify the file that you want to extract as it is stored in the procedure library:

```
prolib $DLC/src/prodoc.pl -extract prodoc/proghand/p-wrk-1.p
```

PROLIB extracts p-wrk-1.p into prodoc/proghand.

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 Client Deployment Guide

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

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.

SQL-92 Reference

(These are non-Progress resources available from your technical bookseller.)

A Guide to the SQL Standard

Date, C.J., with Hugh Darwen. 1997. Reading, MA: Addison Wesley.

Understanding the New SQL: A Complete Guide

Melton, Jim (Digital Equipment Corporation) and Alan R. Simon. 1993. San Francisco: Morgan Kaufmann Publishers.

1

Introduction

The Developer's Toolkit is a product that helps you deploy applications written in the Progress 4GL. It contains several utilities, procedures, and scripts to help you prepare Progress applications for wider distribution. The Developer's Toolkit documentation describes:

- Different Progress 4GL code formats and how they affect deployment
- Upgrading applications
- Specific deployment tasks, such as encrypting source code
- Utilities, procedures, and scripts provided with the Developer's Toolkit

1.1 Toolkit Components

You can use the following set of Developer's Toolkit components when deploying Progress applications:

- DBRSTRCT utility

  Use the DBRSTRCT utility to specify the kinds of access the user has to an application database.

- MKDUMP utility

  Use the MKDUMP utility to create a dump/reload facility.

- XCODE utility

  Use the XCODE utility to encrypt your source procedures for shipment to the user. Users can compile your procedures against their existing databases without the ability to read the contents of the procedures.

- BUNDLE utility

  Use the BUNDLE utility to put Progress binary files into a file called a bundle, then transport the bundle to another machine using FTP, tape, or some other mechanism, and extract the files from the bundle.

- Toolkit templates

  Use these templates to perform various tasks related to deploying Progress applications.

See Chapter 5, "Deployment Topics and Tasks," and Appendix A, "Toolkit Contents and Reference," for more information about these components.

2

Choosing a Code Format

Choosing a code format is the fundamental decision that you must make about deploying an application. There are three different types of code formats:

- Unencrypted source
- Encrypted source
- R-code (CRC-based and time-stamp-based)

This chapter describes each type of code format, the Progress product requirements for those code formats, and the advantages and disadvantages of each format.

2.1 Unencrypted Source

Unencrypted source is uncompiled source that users can read and edit with any standard text editor. Users with full compilation capabilities can read and modify unencrypted source. Thus, when you choose to deploy using unencrypted source, you give users unrestricted access to your application code and application database.

The following sections provide information about the requirements for using unencrypted source, and discuss some advantages and possible disadvantages to deploying unencrypted source.

2.1.1 Development Product Requirements

Developers must be able to repeatedly modify, compile, and test your application to ensure that your application behaves as expected. You must therefore have a full development Progress product on your development platforms. The 4GL Development System, ProVISION, ProVISION Plus, and WebSpeed products are all full development products.

2.1.2 User Product Requirements

To read, modify, compile, and execute unencrypted source, the user must have a full development Progress product (that is, the 4GL Development System, ProVISION, ProVISION Plus, or WebSpeed). The Query/Results Client product is not a full development product because it can compile only read-only application files (source files that contain no code that modifies the database). Also, products that contain client execution engines—for example, the Database Server or Client Networking products—cannot compile unencrypted source.

2.1.3 Advantages of Unencrypted Source

Unencrypted source offers several advantages over other code formats:

- Application extendibility

  Since users have full access to your source code and full compilation capabilities, they can modify your application code any way they want.

- Less to manage at development site

  Other code formats require you to manage multiple code trees and might require you to keep multiple copies of your application database.

- Simpler initial deployment

  When you choose unencrypted source, deploying your application is usually easier than with other code formats. Chapter 3, "Initial Deployment," describes the steps required for each code format.

- Simpler upgrades

  When you choose unencrypted source, upgrading your application is usually less complex than with other code formats. Chapter 4, "Upgrades," describes the steps required for each code format.

- Easier maintenance

  Since users have full development environments, it is relatively easy to modify or enhance your application at their sites.

- Full portability

  Both unencrypted and encrypted source are completely portable across platforms and user interfaces (character mode and GUI), provided you have followed the necessary guidelines for writing portable applications. This is not true for r-code, which is not portable across user interfaces.

- Less space used on storage media

  Unencrypted source files usually occupy much less storage space than r-code files, decreasing their physical media shipment costs. (Encrypted source files also have this advantage over r-code files.)

2.1.4 Possible Drawbacks of Unencrypted Source

Unencrypted source has several possible drawbacks:

- Does not prevent code modifications

  Users can modify the application source, which can potentially cause difficulty when applying application upgrades.

- Does not provide application source code protection

  Unencrypted source does not protect your application from the user. Many Independent Software Vendors (ISVs) do not want the user to be able to see their code.

- Does not prevent database schema modifications

  Many ISVs do not want users to modify their application database. If you deploy using unencrypted source, you lose the ability to prevent unwanted changes. Users can modify your application and application database any way they want.

- More difficult internationalization

  Only r-code can contain multiple text segments for different spoken languages. For more information, see the *Progress Translation Manager Guide*.

- More difficult to support

  Users might create bugs when they modify the source code.

2.2 Encrypted Source

Encrypted source is source code that is encrypted with the XCODE utility (provided with the Developer's Toolkit). The user cannot make changes to encrypted source code because it is not readable by the user, so encrypting the source code protects your application.

The following sections provide information about the requirements for using encrypted source and discuss some advantages and possible disadvantages to deploying encrypted source.

2.2.1 Development Product Requirements

The requirements for encrypted source are the same as for unencrypted source. However, you also need one copy of the Developer's Toolkit (which contains the XCODE utility) for the development platform. You do not need a copy of XCODE for the different target platforms.

2.2.2 User Product Requirements

Any Progress client, server, or standalone database product can compile and run encrypted source.

2.2.3 Advantages of Encrypted Source

Encrypted source code has several possible advantages over other code formats:

- Application protection

  Because encrypted source cannot be read by users, it protects your application from users by preventing them from making changes. Encrypted source shares this advantage with r-code, which is also not user-readable.

- Less to manage at development site

  Encrypting source code requires storing two code trees: the unencrypted source tree and the encrypted source tree (which must be kept in a parallel directory structure). While this is twice what is required with unencrypted source, it is still less than what is required for r-code.

- Full portability

  Like unencrypted source, encrypted source is completely portable across platforms and user interfaces (character mode and GUI). This is not true of r-code, which is not portable across user interfaces.

- Compile-time flexibility

  Because encrypted code is not precompiled, your application can take advantage of compile-time functionality. For example, you might want to pass arguments to include files to determine sort order, field lists, etc. You can take advantage of this behavior at the user site, allowing users to specify their preferences before compilation. This option is not available with r-code, which you must compile at the development site.

- Flexible index definitions

  If users have full development Progress products, they can modify their indexes without invalidating your application code (provided that you have not explicitly named indexes in your code). This flexibility is not available with r-code. R-code that specifically references a new or changed index will not work. Because the index Cyclic Redundancy Check (CRC) value is checked at run time, not all r-code will be affected. Time-stamp-based r-code is even more restrictive.

- Less space used on storage media

  Encrypted source files usually occupy much less storage space than r-code files, making your physical media shipment smaller and less expensive. (Unencrypted source files also have this advantage over r-code files.)

- Encryption keys

  Encrypted source allows you to specify different encryption keys for different code trees or subsystems within a code tree. Therefore, it is possible to deploy an entire application at once, but only give the user encryption keys for certain parts of the application. This way, you can upgrade the user's application without physically distributing code a second time.

2.2.4 Possible Drawbacks of Encrypted Source

Encrypted source code has a number of possible drawbacks:

- Does not prevent index modifications

  When compared to r-code, encrypted source is less restrictive because it allows modifications to the database's indexes. Whether this is an advantage or disadvantage depends on whether you want to give the user the flexibility to change indexes. If your code explicitly specifies indexes in its search criteria, user modifications to your indexes might invalidate your code.

- The Encrypted Compiler Mode (-rx) parameter changes the capabilities of the Compiler

  When you start Progress with the -rx parameter, the Progress Compiler can compile only encrypted source, not unencrypted source. If users have the 4GL Development System, ProVISION, ProVISION Plus, or WebSpeed, they must start Progress again without the -rx parameter to write and compile their own procedures.

2.3 R-code

R-code, or run-time code, is precompiled code. When you compile a source procedure, Progress places the compiled version of the procedure into a file that has the same name as the source procedure with a .r extension. There are two types of r-code: time-stamp-based and CRC-based. For a complete discussion of these two types of r-code, see the *Progress Programming Handbook*.

Like encrypted source, r-code is not readable to the human eye and therefore protects your application from users attempting to make changes.

2.3.1 Developer Product Requirements and R-code Portability

R-code is portable between any two environments if the following are true:

- The user interface is the same—MS-Windows or character mode.

  NOTE: If a procedure contains no code that writes to the screen, its r-code is portable across user interfaces. This is typical, for example, of database schema trigger procedures.

- The DataServer or combination of DataServers connected during compilation is the same.

All machines that require 4-byte or less alignment are compatible. Thus, r-code compiled on one of these machines (a Sun SPARCstation, for example) can run on most other machines. One of the few machines not compatible with this class is the AS/400, which requires greater than 4-byte alignment.

Thus, you can use the same r-code compiled for a character interface and the ORACLE DataServer on any compatible machine (for example, Sun SPARCstation and RS-6000) that also accesses the ORACLE DataServer. Similarly, you can use the same r-code compiled in Windows on any DataServer-compatible Windows system.

You must be able to reproduce the target environment to generate the required r-code. For each combination, you have to keep track of a different code tree. For each combination, you also require a full development Progress product (the 4GL Development System, ProVISION, ProVISION Plus, or WebSpeed) to do the compilations.

2.3.2 User Product Requirements

Any Progress client, server, or standalone database product can run r-code.

2.3.3 Advantages of R-code

R-code provides several advantages over other code formats:

- Speed

  This format is faster because the R-code does not have to compile at run time.

- Application protection

  Like encrypted source, r-code protects your application from possible user changes.

- Good for internationalization

  R-code is the only code format that can contain multiple text segments. This means that the same application can contain strings for multiple spoken languages (French and German, for example). For more information about translating Progress applications, see the *Progress Translation Manager Guide*.

- Prevents database modifications

  Time-stamp-based r-code is tightly coupled to the application database because each r-code file contains a time stamp for each database table that it accesses. If the time stamps do not match, then the r-code does not run. Modifying the database schema—by deleting a file, or changing a field's name, or modifying an index—changes the table's time stamp. Users are therefore very limited in what changes they can make to the database without invalidating the application.

  CRC-based r-code is less tightly coupled to the application database than time-stamp-based r-code. However, CRC-based r-code is still more tightly coupled to the database than encrypted or unencrypted source. For each table in the database, Progress calculates a CRC value that represents the structure of the table. Like a time stamp, a CRC value ensures that the tables in the database match those in the r-code. However, because a CRC value is not specific to a particular database, it is more flexible than a time stamp.

  With either r-code format, you can further restrict access to your application database with the DBRSTRCT utility. This procedure restricts everyone's access to the database, including your own. After you run this procedure, no one can write or compile code that updates the database or changes the database's schema. The only code that can access the database is r-code that you compiled before running the DBRSTRCT utility. This level of database protection is available only with r-code. For more information about the DBRSTRCT utility, see the "Restricting Database Access" section in Chapter 5, "Deployment Topics and Tasks."

2.3.4 Possible Drawbacks of R-code

R-code has several possible drawbacks:

- Limited portability across user interfaces

  The "Developer Product Requirements and R-code Portability" section earlier in this chapter describes the limitations of r-code portability. Depending on the number of incompatible environments requiring a separate compilation, you might have to keep track of multiple r-code trees.

- Loss of compile-time flexibility

  Because you deliver your application in a precompiled format, you cannot make use of compile-time functionality. For example, you cannot pass arguments to include files to determine sort order, field lists, etc. Users therefore cannot specify their preferences before compilation.

- More to manage at the development site

  R-code's limited portability often requires you to manage multiple code trees for a single version of a single application. As you modify or fix your application and create new versions, the number of code trees can become quite large.

  R-code is also tightly coupled with the application database. This tight coupling requires you to keep a copy of all of your application databases that have different CRC values or different time stamps (depending on which type of r-code you have). After numerous upgrades and fixes, the number of databases can grow quite large.

- Time-stamp-based r-code requires a copy of new database for upgrades

  When you modify the schema of your application database, you invalidate the existing r-code because you change the database's time stamps. To deploy an upgrade, therefore, you must provide users with a new database (with the correct time stamps) for each new version of the application. Users are then required to dump their old data and load it into the new database. This is a time consuming process for larger databases. For very large databases, the time required to perform a dump and load can be unacceptable. Progress Software recommends that you use CRC checked sums instead of time stamps for more flexibility.

3

Initial Deployment

This chapter provides an overview of the steps that are required to deploy a Progress application using different code formats. It does not cover every possible contingency, but rather tries to give you a general understanding of the initial deployment process for the different code formats. This chapter puts more emphasis on what to do, rather than how to do it.

These steps are not meant to be performed in a strictly chronological order. More detailed information about particular aspects of deployment is provided inChapter 5, "Deployment Topics and Tasks."

Figure 3–1 illustrates the steps required and options available when deploying new database structures and application procedures. When deploying a database, Progress Software recommends that you first create an empty database at the deployment site, then load the .df files to create the database structure. This method of deployment will help you avoid any difficulties resulting from incompatibility between platforms, operating systems, or Progress versions at the development and deployment sites.

When deploying application procedures, you can choose to deploy either unencrypted or encrypted source code, or to deploy compiled r-code. An important consideration in which method you choose is whether there is a full development product (4GL Development System, ProVISION, ProVISION Plus, or WebSpeed) capable of compiling code at the deployment site. Users with Progress products that cannot compile unencrypted source code will have to deploy encrypted source code and compile it at the site, or deploy r-code. If you deploy r-code, you must ensure that the platforms, operating systems, and Progress versions at the deployment site match those at the development site. Otherwise, you will have to compile the r-code at the deployment site to ensure compatibility. The following sections address these considerations in more detail.

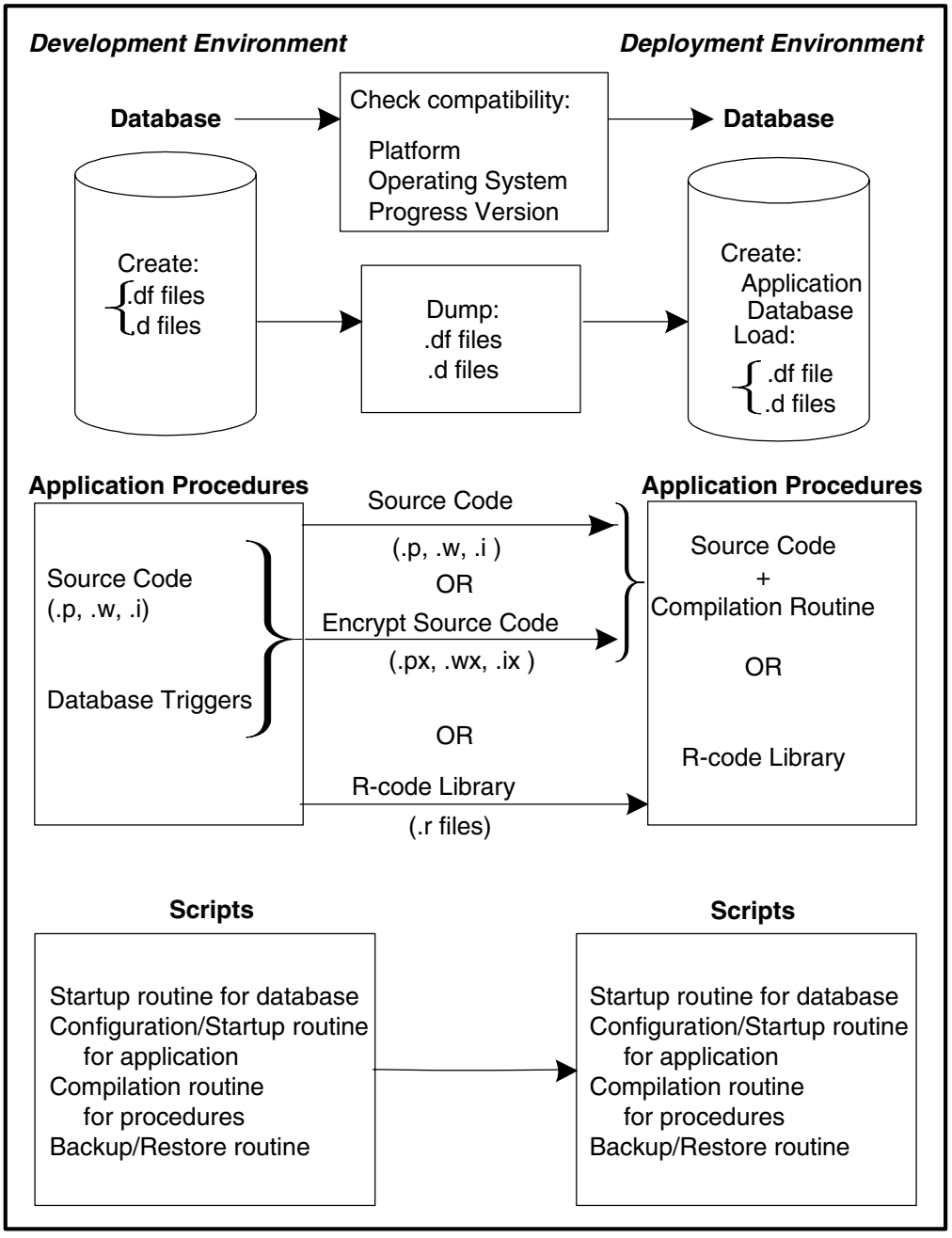


Figure 3–1: Deploying Database Structures and Application Procedures

3.1 Unencrypted Source

This section provides an overview of the steps to follow to deploy an application using unencrypted source when you are done developing and testing your application.

- √ **Dump .df and .d Files**

You must dump your application schema (.df) and/or data (.d) into ASCII format. This step copies your data from the application database, which is not portable, into a format that is portable.

Keep in mind that the .d and .df files should be represented with the appropriate code page for the target platform. For more information about character processing in Progress, see the *Progress Client Deployment Guide*.

- √ **Transfer Files**

You must transfer the resultant .d and .df files and your unencrypted source procedures (which also must be in the appropriate code page) to the target platform. This is a preliminary step to creating your distribution media.

There is more than one way to transfer files. For example, you can use FTP or other protocols, such as CROSSTALK. Or you can put the files on tape or other storage media and physically transport the files.

- √ **Rebuild Application Database at User Site**

You must provide a way for users to build the application database at their site. To do this, the users must be able to load your .df and .d files into an empty database.

Because you are deploying unencrypted source, users must have either the 4GL Development System, ProVISION, ProVISION Plus, or WebSpeed products. These products contain a complete Data Dictionary, which users can use to load the .df and .d files into an empty database. (In character environments, the Data Dictionary contains data administration capabilities. In Windows, these capabilities are provided with the Data Administration tool.)

However, even though the user can perform these tasks, usually it is a good idea to write a script that automates this process. This not only frees the user from performing this task, but also makes your application easier to install. When you buy a full development Progress product, you get all of the Data Dictionary procedures in the `DLC/src/product` directory. The procedures that load data definitions and data are called `load_df.p` and `load_d.p` respectively. Within your script, you can start Progress and run these procedures to load your data definitions and data.

If you decide not to write a script and instead rely on the users' ability to load the data definitions and data, make sure you provide the users with adequate documentation to guide them through the process.

- √ **Copy an Empty Database and Rename It**

To rebuild your application database at the user's site, you must make a copy of the empty database and rename it. An empty database is included in the `DLC` directory at the user's site. Because databases are not portable, you must use an empty database that already is ported to the target platform.

Users can copy their empty database and rename it. Or you can write a script that does this task for them. This script can call the PRODB command to make a copy of the database and rename it appropriately. The Developer's Toolkit also provides a newdb script that creates a database. For more information about this script, see Appendix A, "Toolkit Contents and Reference."

If you decide to let the users perform this step, you must provide adequate documentation to guide them through the process.

- √ **Provide Install Utility**

You must provide a way for users to install your application. Progress must be installed before they can use your application. You can create one install for both Progress and your application. To do this, create an install program using a third-party tool, such as Installshield, and call the Progress Install batch mode from within it.

- √ **Provide Compilation Routine**

This step is optional because users must have either the 4GL Development System, ProVISION, ProVISION Plus, or WebSpeed products, which can compile your application on the fly. However, an application that is precompiled runs faster than an application that is compiled at run time. Therefore, you might want to provide a routine that compiles all of your procedures before it is run.

You can provide a script to do the compilations, or you can provide the user with adequate documentation to do this task on their own. The Developer's Toolkit provides a template procedure, `compile.p` that you can use to compile your application.

- √ **Provide Startup Routine**

Provide a way for users to start the application. You can either write a script to do this or provide adequate documentation.

- √ **Provide Documentation**

You must provide users with enough documentation to use your application and to complete the above tasks if you do not automate them.

- √ **Provide Online Help**

You might want to provide online help for your application. For more information, see the *Progress Help Development Guide*.

- √ **Prepare Distribution Media**

You must prepare your distribution media. It should include:

- All procedures and include files
- All .df and .d files
- All scripts
- All bitmap files, if any
- All DLLs, if deploying in MS-Windows
- All help files
- All online documentation

√ **Now You Are Ready to Deploy Your Application**

3.2 Encrypted Source

This section provides an overview of the steps to follow to deploy an application using encrypted source when you are done developing and testing your application.

- √ **Encrypt Your Procedures and Include Files**

Before you can deploy your application, you must encrypt all of your procedures and include files with the XCODE utility. For more information about how to do this, see the "Encrypting Procedures" section in Chapter 5, "Deployment Topics and Tasks." Before encrypting the files, make sure they are represented with the appropriate code page for the target platform. For more information about character processing in Progress, see the *Progress Client Deployment Guide*.

- √ **Dump .df and .d Files**

As with unencrypted source, you must dump your application schema (.df) and/or data (.d) files into ASCII format. This step copies your data from the application database, which is not portable, into a format that is portable. You do not have to encrypt your .df and .d files.

Like your procedure files and include files, the .d and .df files must be represented with the appropriate code page for the target platform.

- √ **Transfer Text Files**

You must transfer the resultant .d and .df files to the target platform. This is a preliminary step to creating your distribution media.

There is more than one way to transfer files. For example, you can use FTP or other protocols, such as CROSSTALK. Or you can put the files on tape or other storage media and physically transport the files.

- √ **Transfer Binary Files (Encrypted Files)**

After you encrypt procedures and include files, they become binary files.

- √ **Rebuild Application Database at User Site**

You must provide a way for users to build the application database at their site. To do this, users must be able to load your .df and .d files into an empty database.

Any Progress client, server, or stand-alone database product can compile and run encrypted source. However, only those products that can fully compile can load .d files into a database. The 4GL Development System, ProVISION, ProVISON Plus, and WebSpeed are the only products with this capability.

If you deploy to users whose licensed products do not have this capability, you must provide a way for them to load your .d files. For more information about providing a dump/load facility for users, see the "Dumping and Loading" section in Chapter 5, "Deployment Topics and Tasks."

NOTE: The dump/load procedures should be considered part of your application. The decision to distribute encrypted source versions of your application procedures applies to the dump/load procedures as well.

The 4GL Development System, ProVISION, ProVISON Plus, and WebSpeed products contain a complete Data Dictionary, which users can use to load the .df and .d files into an empty database. (In character environments, the Data Dictionary contains data administration capabilities. In Windows, these capabilities are provided with the Data Administration tool.) However, even though the user can perform these tasks, usually it is a good idea to write a script

that automates this process. This not only frees the user from performing this task, but also makes your application easier to use.

Within your script, you can use pre-existing procedures in the DLC/tty and DLC/gui directories (installed for full development Progress products). Within these directories are procedures that load .df and .d files, called load_df.p and load_d.p, respectively. Your script can start Progress and run these procedures to load your data definitions and data. To run these procedures at the user site in the case where users do not have a license for the 4GL Development System, ProVISION, or WebSpeed, the procedures must be encrypted or in r-code format. This step is necessary because Progress must be started with the -rx parameter to run or compile the procedures where users do not have a license. When started with this startup parameter, Progress cannot run or compile unencrypted source.

If you decide not to write a script and instead rely on the users' ability to load the data definitions and data, make sure you provide adequate documentation to guide them through the process.

- √ **Copy an Empty Database and Rename It**

To rebuild your application database at the user's site, you must make a copy of the empty database and rename it. An empty database is included in the DLC directory of all Progress client, server, or stand-alone database products. Because databases are not portable, you must use an empty database that already is ported to the target platform.

Users can use Progress to copy the empty database and rename it. Or you can write a script that does this task for them. This script can call the PRODB command to make a copy of the database and rename it appropriately. The Developer's Toolkit also provides a newdb script that creates a database.

If users have a Progress product that provide these capabilities, and you decide to let them perform this step, you must provide them with adequate documentation to guide them through the process.

- √ **Provide Install Utility**

You must provide a way for users to install your application. Progress must be installed before they can use your application. You can create one install for both Progress and your application. To do this, create an install program using a third-party tool, such as Installshield, and call the Progress Install batch mode from within it.

- √ **Provide Compilation Routine**

When you deploy with encrypted source, you typically provide a compilation routine. Though this is not necessary in all cases, it is usually a good idea because it frees the user from performing the task of compiling your application. In the case where you specify unique encryption keys, providing a compilation routine becomes a necessity.

All Progress client, server, and stand-alone database products can compile encrypted source. Therefore, if you use the default encryption key, users can compile your application without a specialized routine. However, if you have users compile your application, you must provide adequate documentation to guide them through the process.

Because encrypted code is not precompiled, your application can take advantage of compile-time functionality. For example, you might want to pass arguments to include files to determine sort order, field lists, etc. You can take advantage of this behavior at the user site, allowing users to specify their preferences before compilation. This option is not available with r-code, which must be compiled at the development site. Whether you can repeatedly take advantage of compile-time features during the ordinary day-to-day operation of your application depends on what Progress products the users have.

To compile encrypted source code, Progress must be started with the -rx startup parameter. When a Progress session is running with this startup parameter, the Progress Compiler can compile encrypted source but cannot compile unencrypted source. If users have a full development product (the 4GL Development System, ProVISION, ProVISION Plus, or WebSpeed), they will probably want to write and compile their own unencrypted procedures. Thus, in their day-to-day operation of Progress, they must run Progress **without** the -rx startup parameter. For users with full development products, therefore, you probably want to perform a one-time compilation with the -rx parameter, then restart Progress without this parameter. For users with Progress products that cannot compile ordinary unencrypted source, running permanently with the -rx parameter is not an issue.

The Developer's Toolkit provides a template procedure, `compile.p`, that you can use to compile your application. For more information about `compile.p`, see the "Decryption and Compilation" section in Chapter 5, "Deployment Topics and Tasks."

- √ **Provide Startup Routine**

You must provide a way for users to start the application. You can either write a script to do this or provide adequate documentation.

- √ **Provide Documentation**

You must provide users with enough documentation to use your application and to perform the above tasks if you do not automate them.

- √ **Provide Online Help**

You might want to provide online help for your application. For more information, see the *Progress Help Development Guide*.

- √ **Prepare Distribution Media**

Now that you have completed the above steps, you need to prepare your distribution media. It should include:

- All procedures and include files (encrypted)
- All data definition (.df) and data (.d) files
- All scripts
- All bitmap files, if any
- All DLLs, if deploying on MS-Windows
- All help files
- All online documentation

√ **Now You Are Ready to Deploy Your Application**

3.3 CRC R-code

This section provides an overview of the steps to follow to deploy an application using CRC-based r-code. When you are done developing and testing your application and are ready to begin deploying, follow the steps described in the following sections. For a complete description of CRC-based r-code, see the *Progress Programming Handbook*.

- √ **Decide Whether to Restrict Database Access**

If you deploy your application to users with full development products (the 4GL Development System, ProVISION, or ProVISION Plus), they can write their own applications to access the database and modify the database schema. Depending on the nature of your application, you might not want this to happen. To restrict access to the database, you can use the DBRSTRCT utility that comes with the Developer's Toolkit. For more information about using this procedure, see the "Restricting Database Access" section in Chapter 5, "Deployment Topics and Tasks."

If you decide to restrict access to the database, you must physically deploy a copy of your application database with your application. It is not possible for users to build your database at their site. Because databases are not portable, you must generate a copy of the database for each target platform. Then you must run the DBRSTRCT utility on each platform against all of the databases.

In general, restricting access to the database makes it more difficult to initially deploy and upgrade applications, but it also helps minimize user-created errors. For each release of the application, the development site must:

- Generate a database for each platform
- Compile the application against each database
- Save copies of each database and each r-code tree

To update the application, the user must dump the existing database and load the data into a new database that you supply. For users with large mission-critical databases, this might be an unacceptable requirement.

- √ **Compile Your Procedures and Include Files**

As described in Chapter 2, "Choosing a Code Format," r-code is not completely portable. For each combination of factors requiring a separate compilation, you need to perform a separate compilation and keep track of a different code tree at the development site. For each combination, you also need a full development Progress product (the 4GL Development System, ProVISION, or ProVISION Plus)) to do the compilations.

Unless you explicitly specify the Time Stamp (-tstamp) startup parameter when you start Progress, Progress generates CRC-based r-code files at compilation.

NOTE: Progress Software recommends CRC-based r-code because it affords greater flexibility.

- √ **Dump .df and .d Files**

If you decide not to restrict database access, you should dump your application schema (.df) and/or data (.d) files into ASCII format. This step copies your data from the application database, which is not portable, into a format that is portable.

The .d and .df files should be represented with the appropriate code page for the target platform. For more information about character processing in Progress, see the *Progress Client Deployment Guide*.

- √ **Transfer Text Files**

You must transfer the resultant .d and .df files to the target platform.

There is more than one way to transfer files. For example, you can use FTP or other protocols, such as CROSSTALK. Or you can put the files on tape or other storage media and physically transport the files.

- √ **Transfer R-code Files**

To aid in moving r-code files between platforms, you can use the BUNDLE utility. For more information about the BUNDLE utility, see Chapter 5, "Deployment Topics and Tasks."

- √ **Rebuild Application Database at User Site**

If you have decided not to restrict access to the database, then you must provide a way for users to rebuild your application database at their site. To do this, users must be able to load your .df and .d files into an empty database.

NOTE: You can restrict access to the database and provide compiled code to load .d files.

Any Progress client, server, or stand-alone database product can run r-code. However, not all of these products can load .d files into a database. The 4GL Development System, ProVISION, and ProVISION Plus are the only products that have this capability.

If you deploy to users whose licensed products do not have this capability, you must provide a way for them to load your .d files. For more information about providing a dump/load facility for users, see the "Dumping and Loading" section in Chapter 5, "Deployment Topics and Tasks."

NOTE: The dump/load procedures should be considered part of your application. The decision to distribute r-code versions of your application procedures applies to the dump/load procedures as well.

The 4GL Development System, ProVISION, or ProVISION Plus products contain a complete Data Dictionary, which users can use to load the .df and .d files into an empty database. (In character environments, the Data Dictionary contains data administration capabilities. In Windows, these capabilities are provided with the Data Administration tool.) However, even though the user can perform these tasks, usually it is a good idea to write a script that automates this process. This not only frees the user from performing this task, but also makes your application easier to use.

Within your script, you can use pre-existing procedures in the `DLC/tty` and `DLC/gui` directories (installed for full development Progress products). Within these directories are procedures that load .df and .d files, called `load_df.p` and `load_d.p`, respectively. Your script can start Progress and run these procedures to load your data definitions and data. To run these procedures at user sites where users do not have a license for the 4GL Development System or ProVision, the procedures must be in r-code format.

If you decide not to write a script and instead rely on the users' ability to load the data definitions and data, make sure you provide adequate documentation to guide them through the process.

- √ **Copy an Empty Database and Rename It**

To rebuild your application database at the user's site, you must make a copy of the empty database and rename it. An empty database is included in the DLC directory of all Progress client, server, or stand-alone database products. Because databases are not portable, you must use an empty database that already is ported to the target platform.

Users can use Progress to copy the empty database and rename it. Or you can free them from this responsibility by writing a script that does this task for them. This script can call the PRODB command to make a copy of the database and rename it appropriately. The Developer's Toolkit also provides a newdb script that creates a database.

If users have a Progress product that provide these capabilities, and you decide to let them perform this step, you must provide adequate documentation to guide them through the process. For example, you should let them know there are multiple empty databases, and you should recommend which one they should use.

- √ **Provide Install Utility**

You must provide a way for the user to install your application. Progress must be installed before they can use your application. You can create one install for both Progress and your application. To do this, create an install program using a third-party tool, such as Installshield, and call the Progress Install batch mode from within it.

- √ **Provide Startup Routine**

You must provide a way for the user to start the application. You can write a script to do this or provide adequate documentation.

- √ **Provide Documentation**

You must provide the user with enough documentation to use your application and to perform the above tasks if you do not automate them.

- √ **Provide Online Help**

You might want to provide online help for your application. For more information, see the *Progress Help Development Guide*.

- **Prepare Distribution Media**

Now that you have completed the above steps, you need to prepare your distribution media. It should include:

- All compiled procedures
- All application procedures
- All dump and load procedures
- All data definition (.df) and data (.d) files
- All scripts and bitmap files, if any
- All DLLs, if deploying in MS-Windows
- All help files
- All online documentation

3.4 Time-stamp R-code

This section provides an overview of the steps to follow to deploy an application using time-stamp-based r-code when you are done developing and testing your application. For a complete discussion of time-stamp-based r-code, see the *Progress Programming Handbook*.

- √ **Decide Whether to Restrict Database Access**

If you deploy your application to users with full development Progress products (the 4GL Development System, ProVISION, or ProVISION Plus), the users can write their own applications to both access and modify the application database. Depending on the nature of your application, you might not want this to happen. To restrict access to the database, you can use the DBRSTRCT utility that comes with the Developer's Toolkit. For more information about using this procedure, see the "Restricting Database Access" section in Chapter 5, "Deployment Topics and Tasks."

If you decide to restrict access to the database, you must physically deploy a copy of your application database with your application, because it is not possible for users to build your database at their site. Databases are not portable, so you must generate a copy of the database for each target platform. Then you must run the DBRSTRCT utility on each platform against all of the databases.

In general, restricting access to the database makes it more difficult to initially deploy applications. The development site must:

- Generate a database for each platform
- Compile the application against each database
- Save copies of each database and each r-code tree

Restricting the database removes any advantage that the portability of r-code gives you. However, it does ensure that the database schema cannot be modified.

- √ **Build Database for Each Target Platform**

Time-stamp-based r-code is closely coupled with the application database: both the database time stamps and the r-code time stamps must match for the application to run. It is therefore necessary to build the database on each target platform before you compile your application. To build the database, you can copy an empty database to your target platform, then load a .df file. You can also load .d files into the database.

After you create r-code for each target platform, you need to duplicate the r-code and database as a preliminary step to creating your distribution media. At your development site, you must keep a copy of the application database for each target platform.

- √ **Compile Your Procedures and Include Files**

As described in Chapter 2, "Choosing a Code Format," r-code is not completely portable. For each combination of factors requiring a separate compilation, you have to generate and keep track of a different code tree at the development site. To do the compilations, you also need a full development Progress product (the 4GL Development System, ProVISION, or ProVISION Plus). Once you have generated the necessary code trees, you can duplicate the r-code when you create your distribution media.

Unless you explicitly specify the -tstamp parameter when you start Progress, Progress generates CRC-based r-code files at compilation.

- √ **Transfer Text Files**

You must transfer any .d files that are part of your application to the target platform. This is a preliminary step to creating your distribution media, which must be created on the target platform.

There is more than one way to transfer files. For example, you can use FTP or other protocols, such as CROSSTALK. You can also put the files on tape or other storage media and physically transport the files.

- √ **Transfer R-code Files**

To aid in moving r-code files between platforms, you can use the BUNDLE utility. For more information about the BUNDLE utility, see the "BUNDLE Utility" section in Chapter 5, "Deployment Topics and Tasks."

- √ **Provide Dump/Load Facility**

If users have the 4GL Development System, ProVISION, or ProVISION Plus products, they have full dump/load capabilities. If users do not have these products, you must provide a dump/load facility. If the database is restricted, you must supply r-code files to dump and load. For more information, see the"Dumping and Loading" section inChapter 5, "Deployment Topics and Tasks."

NOTE: The dump/load procedures should be considered part of your application. The decision to distribute r-code versions of your application procedures applies to the dump/load procedures as well.

- √ **Provide Install Utility**

You must provide a way for the users to install your application. If they do not have a Progress product installed, they must install Progress before they can use your application. You can create one install for both Progress and your application. To do this, create an install program using a third-party tool, such as Installshield, and call the Progress Install batch mode from within it.

- √ **Provide Startup Routine**

You must provide a way for users to start the application. You can either write a script to do this or provide adequate documentation.

- √ **Provide Documentation**

You must provide users with enough documentation to use your application and to perform the above tasks if you do not automate them.

- √ **Provide Online Help**

You might want to provide online help for your application. For more information, see the *Progress Help Development Guide*.

- √ **Prepare Distribution Media**

Now you have to prepare your distribution media. It should include:

- Application database
- All procedure and include files (or application r-code if the database is restricted), and dump/load procedures
- All .df (unless the database is restricted) and .d files
- All scripts
- All bitmap files, if any
- All DLLs, if deploying in MS-Windows
- All help files
- All online documentation

4

Upgrades

After you distribute your application and users have been using it, you might want to upgrade, fix, or modify either the application procedures or the application database. It is a good idea to develop a strategy for dealing with application upgrades before distributing an application. That way, when you have to upgrade an application, you are less likely to have unexpected problems. This chapter helps you prepare an effective upgrade strategy; it covers the following topics:

- Distributing upgraded application procedures
- Preparing and distributing changes to the database structure

Modifying an application is relatively straightforward if you have not made changes to the database schema. If you have made changes to the database schema, modifying the application requires more steps.

Figure 4–1 illustrates the steps needed and options available when upgrading and distributing changes to the database structure or application procedures. When making changes to the database structure, you must to create an incremental .df file that contains the changes in the database structure. You can then load this .df file into the existing database structure to update that database. For information on creating an incremental 4GL .df file, see the *Progress Database Administration Guide and Reference*. See the *Progress Programming Handbook* for more information about deploying changes to database structures.

When modifying application procedures, you can choose to deploy either unencrypted or encrypted source code, or to deploy modified r-code. An important consideration in which method you choose is whether there is a full development product (4GL Development System, ProVISION, ProVISION Plus, or WebSpeed) capable of compiling code at the deployment site. When deploying to users with Progress products that cannot compile unencrypted source code, developers will have to deploy encrypted source code and compile it at the site, or deploy r-code. If you deploy r-code, you must ensure that the platforms, operating systems, and Progress versions at the deployment site match those at the development site. Otherwise, you will have to compile the r-code at the deployment site to ensure compatibility. The following sections address these considerations in more detail.

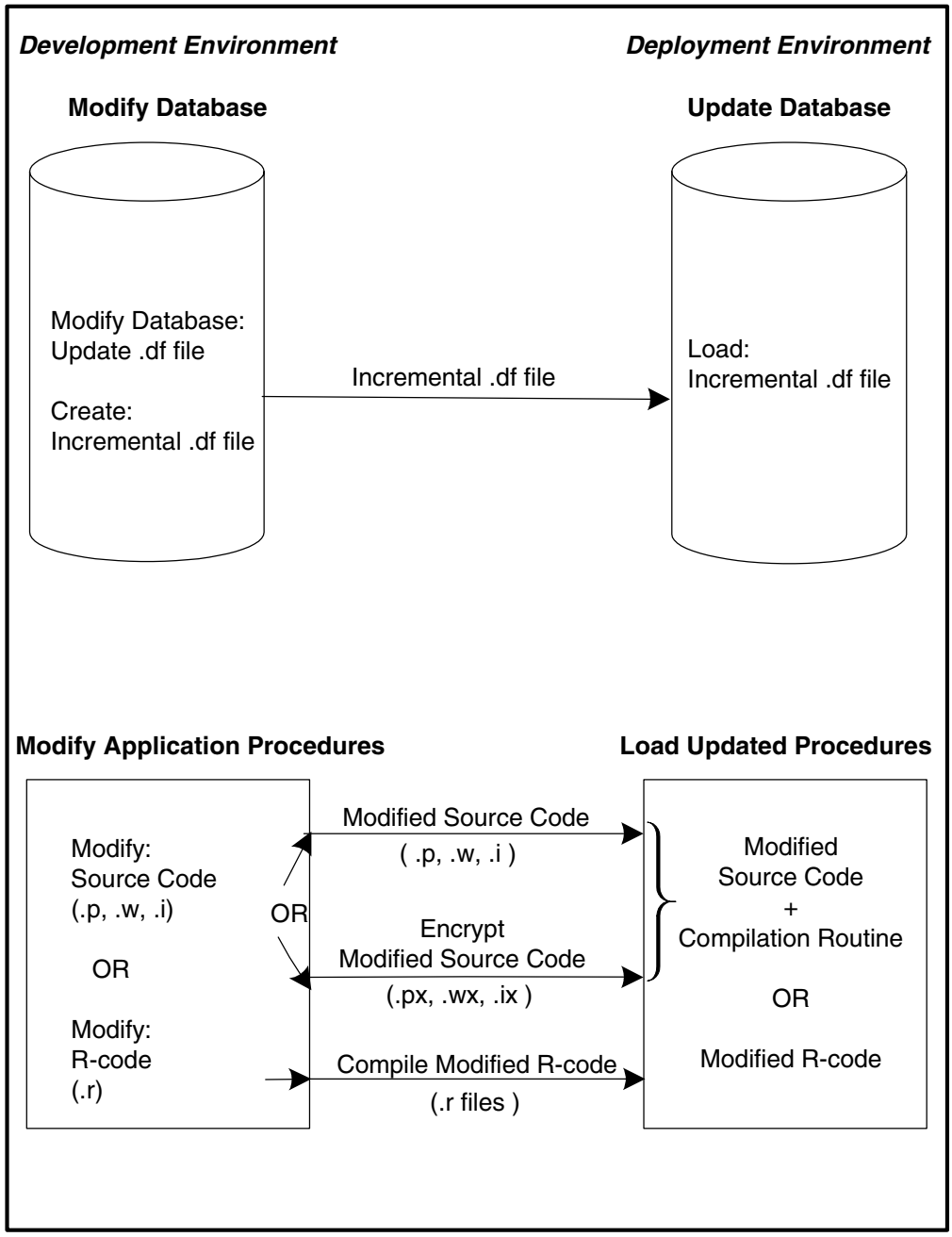


Figure 4–1: Deploying Upgraded Database Structure and Application Procedures

4.1 Modifying the Application Procedures

As with any distribution, you can choose to distribute unencrypted source, encrypted source, or r-code (CRC-based or time-stamp-based). When you distribute r-code, Progress Software recommends you choose CRC-based r-code. This section describes upgrading an application without any modifications to the database. If you have modified the database schema, upgrading becomes more complicated, as described in the "Modifying the Application Database" section.

4.1.1 Unencrypted Source

To upgrade your application, distribute the modified versions of your procedures.

4.1.2 Encrypted Source Files

To upgrade your application, distribute the modified versions of your procedures. If you specify an encryption key, you must also provide a compilation procedure.

As described in Chapter 3, "Initial Deployment," when you deploy with encrypted source, you usually want to provide a compilation procedure. Although this is not necessary in all cases, it is a good idea because it frees the user from having to compile your application. When you specify unique encryption keys, providing a compilation routine is a necessity.

All Progress client, server, and stand-alone database products can compile encrypted source. Therefore, if you use the default encryption key, the user can compile your application without a specialized procedure. However, if you have the user compile your application without a script, you must provide adequate documentation to guide them through the process.

4.1.3 Time-stamp-based R-code

If you are supplying time-stamp-based versions of new or modified procedures, you must compile the procedures against the same version of the database that users are currently using, and on the same machine type. Otherwise, the r-code versions of the new or modified procedures will not run, because database file time-stamps are different at the user site.

Once you have precompiled the procedures against the original database, you can distribute the new r-code files to the user.

4.1.4 CRC-based R-code

If you supply new or modified CRC-based r-code files, you must precompile the procedures against a version of the database that is structurally the same as the user's database. You must also compile it on a machine of the same class, same user interface, and same combinations of DataServers (if any). However, since the r-code you create has no time-stamps, it does not matter when the database tables were created or modified.

4.2 Modifying the Application Database

Upgrading is more complicated when you have to change the database schema. If you are upgrading with unencrypted source, encrypted source, or CRC-based r-code, you can supply an incremental .df file and copies of the modified procedures.

If you are upgrading with time-stamp-based r-code, you must supply the following:

- A new empty application database that includes the modified tables, fields, and indexes
- Precompiled versions of all modified procedures
- Dump/load utilities that will dump the user's data and definitions and load them into the new empty database

4.2.1 Upgrading with Updated .df and Encrypted Source

When you update a database structure in the development environment, you must then re-create these changes in the database that is already established at the deployment site. To accomplish this transfer, an incremental .df file is generated in the development environment. Copies of the original and updated databases are compared, and the differences between them are incorporated into an incremental .df file. For more information on creating an incremental 4GL data definitions file see the *Progress Database Administration Guide and Reference* for more information about deploying changes to database structures. The following general steps tell you how to prepare a .df file that includes new or modified data definitions:

- Obtain copies of the current user database and the new modified database.
- Connect the two databases.
- Compare the data definitions in the user database with the data definitions of your new updated database, creating a new incremental .df file that reflects the changes.
- Provide a utility for users that loads the incremental .df file into the existing database.
- Encrypt the procedures affected by changed data definitions.

In Windows, use the Data Administration Tool and follow these steps (for character interfaces, use the Data Dictionary):

1 ♦ Connect the two databases by choosing Database→ Connect.

2 ♦ Select the database that includes the new, modified data definitions as your current database by choosing Database→ Select Working Database.

3 ♦ Create an incremental definition file by choosing Admin→ Dump Data and Definitions→ Create Incremental .df File.

This option compares the data definitions in the nonempty copy to the current database schema and creates a new data definition (.df) file. The new .df file contains a record for each difference between the two schemas. The differences include any added, renamed, changed, or deleted file, field, or index.

If a file, field, or index exists in the old database but not in the new schema, Progress asks you whether the object has been renamed. If you respond "no," a record appears in the new .df file marking the object as deleted.

If the new schema includes a new unique active index, Progress asks you whether you want to deactivate it. If you do not deactivate the index, and there are duplicate keys in the old database, Progress aborts your attempt to load new definitions into the old database. If you deactivate the index, the load procedure defines the new index, but does not create the index file. You must build and activate the index after loading the new data definitions.

4 ♦ Perform steps a through d below for testing purposes. Then prepare a tool for users that performs these steps on site. (The `upgrade` template provided with the Developer's Toolkit outlines one way to do this. Before you can use this template, you probably need to modify it.)

a) Select the copy of the old database as your working database.

b) Load the updated data definitions by choosing Admin→ Load Data and Definitions→ Load Data Definitions (.df file).

c) If you deactivated any indexes in Step 3, re-create data in the indexed fields as required to avoid duplicate keys. Then reactivate the indexes with PROUTIL IDXBUILD. For more information, see the *Progress Database Administration Guide and Reference*.

d) Progress now updates the old database schema to match the modified schema. Compile and test all your procedures against the updated database.

5 ♦ Test your procedures.

6 ♦ Use the XCODE utility to encrypt procedures invalidated by the new data definitions.

Since the new .df file changes CRC check sums and time-stamps only on the database tables that are actually modified, you only have to encrypt and ship the source versions of those procedures that access the changed tables. (If you compile your procedures with the XREF option, you get a listing of the tables accessed by each procedure.)

Using the Upgrade Template

The upgrade template, all .p files it uses, and all .inp files it uses are only templates; you must check each file and modify it to suit your particular application. Following is a list of checks and modifications you must make before using upgrade; your application might require additional changes.

The UNIX version of upgrade is found in /usr/dlctk/samples/unix. The DOS version is located in the respective subdirectories of samples. The files that upgrade uses can be found in /usr/dlctk/samples.

The upgrade template performs three basic actions to upgrade an application at the user's site.

1. Loads the new .df file, which you created from your version of the upgraded database with the Admin→ Dump Data and Definitions→ Create Incremental .df File option.
2. Rebuilds any indexes deactivated during the data definition load process.
3. Compiles encrypted source procedures.

Changes to the upgrade template for the data definition load stage (Step 1.):

- Modify upgrade.p (not upgrade) and the files it uses:
  - Set the system administrator id and password appropriately in loginupg.p (or prepare a tool your users can use to set them).
  - Modify upgrade.inp to name the new .df file.
  - Test load_df.p (a standard Progress Dictionary procedure). If you have created a new unique active index, load_df.p asks whether you want to deactivate the index. You need to add a response (Y or N) to upgrade.inp.
  - If you and your users do not freeze the database files, delete the call to freeze.p. (Leave the call to unfreeze.p in case your users ever freeze the database files without your knowledge.) See the *Progress Client Deployment Guide* for more information about freezing and unfreezing files.
  - Modify unfreeze.p and freeze.p to include the correct filenames.
  - Check upgrade.log for any problems.

Changes to the upgrade template for the rebuild deactivated indexes stage (Step 2.):

- If you and your users will not be deactivating indexes, remove the _proutil command line from upgrade.
- If you have added a new unique active index, modify rebuild.inp to include the correct responses to prompts by _proutil $1 -C idxbuild. That is, test _proutil idxbuild on your database and add a line to rebuild.inp to answer each question appropriately.
- Check rebuild.log for problems.
- You might want to add a procedure to upgrade that checks rebuild.log and, if necessary, fixes duplicate key data and rebuilds the index again. Alternatively, you might want to check the .log file and just stop before compilation proceeds.

Changes to upgrade template for the compile encrypted procedures stage (Step 3.):

- Modify compile.inp to name the procedures to be compiled.
- Modify the COMPILE statement in compile.p to use the encryption key you want.
- Prepare to ship compile.p: either encrypt compile.p, possibly with the default key, or compile compile.p, thereby hiding your encryption key in the .r file. If you ship compile.r, it must be compiled on the target machine type. It can be compiled against any database because it does not reference any database files.

The upgrade template is relevant when you want to distribute only new data definitions and encrypted versions of affected application procedures. However, you might choose instead to ship a complete new version of the database and application, as described in the following section.

4.2.2 Upgrading with Updated .df and CRC-based R-code

The flexibility of CRC-based r-code makes it easy to distribute a new release of an existing application. Distribution involves the following general steps:

1. Create an incremental .df file of your application database using the Data Administration tool (for graphical environments) or the Data Dictionary's Admin submenu (for character environments). For more information on creating an incremental .df file, see the *Progress Database Administration Guide and Reference*.
2. Recompile source files that were affected by the new schema changes.

3. Send the .df file and copies of the new r-code files that were affected by the schema changes to your users. After the users load the incremental .df file, they can immediately run your new .r code.

Essentially, the steps you take are the same as when you use encrypted source, but you gain the following advantages:

- You do not have to encrypt your procedures. However, you need to recompile some r-code.
- The user does not have to compile your procedures.

In addition, unlike time-stamp-based r-code, you gain these advantages:

- You do not have to send a copy of a new database to the user.
- The user does not have to dump and reload data.

One possible disadvantage of using CRC-based r-code is that a user with Progress 4GL Development System, ProVISION, ProVISION Plus, or WebSpeed can create a counterfeit database (a database with the same structure as your application database). The user can then write a program, compile the program, and run it against your application database. (The user cannot do this if you use time-stamp-based r-code because the time-stamps in the counterfeit code would not match those in your database.)

If this concerns you, the PROUTIL utility's DBAUTHKEY qualifier enables you to set an authorization key for your database. When compiling a procedure, Progress includes the value of this key in your r-code. CRC-based r-code that does not include the correct authorization key will not run against your database.

You can insert the authorization key into existing CRC-based r-code with the PROUTIL utility's RCODEKEY qualifier.

For more information on the PROUTIL utility's DBAUTHKEY and RCODEKEY qualifiers, see the *Progress Database Administration Guide and Reference*.

4.2.3 Upgrading with Time-stamp-based R-code Files

If database changes are extensive, you might choose to distribute brand new versions of your application and the database. If you plan to distribute the precompiled version of an upgraded database, follow these steps.

1 ♦ Build a new basic database.

2 ♦ Recompile **all** your procedures, including those not directly affected by the schema change, so that r-code file time-stamps match the database file time-stamps.

3 ♦ Distribute the new basic database and the recompiled procedures to users.

Your users will have to dump their data from the old database and load it into the new database. Therefore, you must also provide your users with a dump/reload facility. If you are changing any of the file formats used in your application, you must modify the set of load files to work with those new formats.

Once this is done, follow these steps:

1 ♦ Run the mkdump script against the new version of the database.

2 ♦ Supply a modified set of file load procedures so the user can reload the database using procedures that are compatible with the old data format. These procedures will probably be different from the file load procedures the user will use to reload the new database after dumping it.

3 ♦ Supply new file dump, file load, and subdirectory procedures so the user can later dump and reload the database, if necessary.

If you are making database modifications and you are supplying new dump and reload procedures, the user must do the following:

1 ♦ Use the old dump procedure to dump the data in the existing database. (Note that dumping and reloading a large database might take days, depending on the database size.)

NOTE: You can use the binary dump and load to reduce this time dramatically. See the section on PROUTIL in the *Progress Database Administration Guide and Reference* for more information on binary dump and load.

2 ♦ Make a copy of the new basic database.

3 ♦ Use the new load procedure to load the old database data into the new database. You must ensure that the new load procedure is prepared to accept data in the format and order dumped by the old dump procedures.

4.2.4 Progress Product Upgrades

The user might upgrade from a Progress product that does not support full development to either the 4GL Development System, ProVISION, ProVISION Plus, or WebSpeed. If the user's application database had previously been **restricted**, and you now want to allow the user to run your application using the capabilities of their new, less restrictive version of Progress, you must follow these steps:

1 ♦ Create a new basic database.

2 ♦ Distribute the new basic database to the user.

The user must use the dump/load facility you provided to dump the existing database, make a copy of the new basic database, and reload the existing data into that new database.

If the new version of Progress is being installed in a different directory than the old version, UNIX users can use the `_tlr` module to edit the value of the DLC variable in your application scripts.

5

Deployment Topics and Tasks

This chapter describes issues related to deploying Progress applications. Specifically, it discusses:

- Encrypting source code
- Using the BUNDLE utility
- Dumping and loading
- Restricting database access

5.1 Encrypting Procedures

This section describes the XCODE utility and the process of encrypting source code before deploying an application.

5.1.1 XCODE Utility

Use the XCODE utility to encrypt Progress procedures and include files. This is the syntax for XCODE:

SYNTAX (UNIX)

```
xcode [ -k key ] -d directory [ files] [ - ]
```

key

A character string up to 8 characters long. On UNIX, *key* is case sensitive. If you do not specify *key*, XCODE uses a default key. When Progress is started with the Encrypted Compiler Mode (-rx) startup parameter, the Progress Application Compiler automatically uses this default key, unless you specify the XCODE option on the COMPILE statement.

directory

The directory where XCODE places the encrypted *files*. The specified *directory* must be different from the source directory, because each encrypted file retains the name of its original source file. XCODE overwrites existing files in *directory* if they are encrypted.

files

The relative pathnames of the files you want to encrypt. XCODE appends the pathnames supplied to *directory.*

5.1.2 Preparing to Use XCODE

Before running XCODE, create a directory structure exactly parallel to your source directory structure, then move to the root of the original source directory structure. Supply the root of the new structure as *directory*, and supply the relative path names of the files to be encrypted as *files*. The relative path names are appended to *directory.* Because XCODE does not create new directories, any directories implied by the *directory* or *files* arguments must be created before running XCODE. Encrypted files have the same names as the original source files.

You must encrypt both your procedure and include files in order for your encrypted procedures to run.

The dash (-) option tells XCODE to take filenames from the standard input. On UNIX, this allows you pipe output from an `ls`, `cat`, or `find` command to XCODE.

Here is an example of how you can use XCODE on UNIX. Suppose you have just finished developing and testing the procedures in `/usr/test` and its four subdirectories and want to encrypt them for shipment to users. Follow these steps:

1 ♦ Create a directory structure parallel to `/usr/test`:

```
cd /usr/test
mkdir /usr/testx
mkdir /usr/testx/ar
mkdir /usr/testx/fm
mkdir /usr/testx/inv
mkdir /usr/testx/oeFk
```

2 ♦ Move to the original development root in preparation for encryption:

```
cd /usr/test
```

3 ♦ Encrypt your procedures:

```
xcode -k mykey -d usr/testx *.p ar/*.p fm/*.p inv/*.p oe/*.p
```

Encrypted versions of the source procedures in `/usr/test` and its subdirectories now reside, with the same names, in `/usr/testx` and its subdirectories.

NOTE: You must also encrypt include (.i) files.

5.1.3 Decryption and Compilation

If you distribute encrypted source code, you must also prepare and distribute a program that compiles your application procedures at a user site. The core of this program is one or more COMPILE statements with the XCODE option. If you encrypt your procedures using the default key, the XCODE option is unnecessary, because the Compiler recognizes that the procedures are encrypted and attempts to decrypt them using the default key.

The XCODE option of the COMPILE statement decrypts the specified source files-and any encrypted include files-using the same keys supplied in XCODE. Decrypting is incremental during compilation. That is, having the decryption key does not allow a user to examine a decrypted version of the source code. Furthermore, the key can be easily hidden in an object version of your compile program.

When started with the Encrypted Compiler (-rx) startup parameter, Progress places you (or a user) by default in the Progress Procedure Editor. The COMPILE statement and the `icompile.p` procedure will only compile encrypted procedures when Progress is started with -rx. When shipping encrypted source code, you must supply either an encrypted .p or a .r version of your compile program, which can be run in encrypted compiler mode. The Toolkit includes a sample template, called `upgrade`, that invokes Progress with the -rx parameter and compiles encrypted source programs.

You invoke the encrypted source code compiler with the command `PROGRESS/XCOMPILER`. See `upgrade.com` in the Toolkit `samples` subdirectory for more information.

After the procedures are compiled, you might want users to delete the source to regain the disk space. If you do this, consider freezing the database files; otherwise, the user can modify the database and invalidate object procedures for which the user no longer has the source.

5.2 BUNDLE Utility

The BUNDLE utility lets you put Progress binary files into a file called a bundle, transport the bundle to another machine using FTP, KERMIT, tape, or other mechanism, and extract the files from the bundle. The BUNDLE utility is designed for copying encrypted source files and r-code files, but any platform-independent binary file or ASCII file can be copied as well. It is not designed to handle executables, object files, Progress databases, or other platform-dependent files.

The BUNDLE utility consists of two C language programs: `bundle.c` and `unbundle.c`. These programs are described in the following sections.

5.2.1 The bundle.c Program

The bundle program puts input files into a bundle:

SYNTAX (UNIX)

```
bundle [ -ascii | -binary ] [ -uppercase | -lowercase ]
  bundle-file   { { -select [ selection-file ] } | input-file }
```

`-ascii | -binary`

Specifies whether the input file is to be read as an ASCII text file or as a binary file. The default is binary.

`-uppercase | -lowercase`

Tells bundle to convert the names of files being bundled to uppercase or lowercase. By default, filenames are used without case conversion.

`bundle-file`

Provides the name of the file into which input files are bundled. If the file does not exist, bundle creates it. If it does exist, bundle appends data to the end of it.

`-select` *`selection-file`*

Indicates that the names of the input files are to be read from a selection file. By default, the selection file is read from standard input.

This is the format of *selection-file*:

SYNTAX

`filename` **[** `ascii` | `binary` **]**

Each entry must be on a line by itself and cannot continue across lines.

If ascii or binary is absent, the command line specification is used. Use an exclamation point (!) or pound sign (#) as comment characters. Everything from a comment character to the end of line is ignored. The comment character must be separated from any preceding text by a blank space. Blank lines are also ignored. Wild cards are not supported.

This is a sample file:

```
# First line.
# Config files follow.
cfg-show.x   # Depending on command-line, ascii or binary.
cfg-doc.txt  ascii
cfg-make     ascii
# The next line is blank.    # Last line.
```

input-file

The name of a single input file. Wild cards are not supported.

5.2.2 The unbundle.c Program

The unbundle program takes files out of a bundle. This is the syntax:

SYNTAX (UNIX)

```
unbundle [ -toc | -fulltoc | -date | -all ] bundle-file
             [ -select [ selection-file ] ]
```

-toc | -date | -fulltoc | -all

Indicates the operation to perform. The default, toc, lists a table of contents that shows the name and type (ascii or binary) of each file in the bundle. The fulltoc value lists a table of contents containing: 1) toc information, 2) date information, and 3) the input files' file specifications, modify dates, and sizes. The date value lists time stamps showing when bundle was run. The all value unbundles all files in the bundle putting the unbundled files in the current directory.

-select *selection-file*

Allows you to select by name the files to unbundle or to list with toc or fulltoc.

This is the format of *selection-file*:

SYNTAX

```
filename [ unbundle-as-file-spec ]
```

The *filename* value names the file to be processed. When you are unbundling, unbundle-as-file-spec specifies the name unbundle is to use as the unbundled filename. Comments and blank lines are permitted as described earlier for the bundle command's *selection-file*. Wild cards are not supported.

If select is specified and no *specification-file* is provided, selections are read from standard input.

This is a sample file:

```
# UNIX example
# First line.
cfg-show.x   pdir/cfg-show.p
cfg-doc.txt  docdir/cfg-doc.doc
cfg-make     # Created as cfg-make in current directory.
readme.txt   # Created as readme.txt in current dir.
# The next line is blank.

# Last line.
```

5.2.3 Examples Using BUNDLE

The following synopsis shows how to use bundle and unbundle. This section assumes you have to transfer encrypted source files, which are binary files.

First you have to create a "bundle" file. To do this you specify the output file (that is, the bundle) and one or more input files (the encrypted source files). For these examples, the bundle file has a .bun extension and the encrypted source files have a .x extension. The bundle is created on UNIX, copied to Windows, and unbundled on Windows.

Put a single file named `foo.x` in the bundle:

```
# bundle xcode.bun foo.x
```

Or put all encrypted source files in the current directory in the bundle:

```
# ls *.x | bundle xcode.bun -select
```

Or put some selected files (this, that, what) in the bundle:

```
# bundle xcode.bun -select
this.x
that.x
what.x
^D
```

You can also put the selections into a file:

```
# cat >xcode.sel
this.x
that.x
what.x
^D
# bundle xcode.bun -select xcode.sel
```

Note that files are always appended to the bundle, so you can incrementally add files to it. Thus, the above sequence created `xcode.bun` with this in it: `foo.x`, all x-code files in the current directory, `this.x`, `that.x`, and `what.x`.

Now you need to copy the bundle to some destination. You can use FTP, making sure that binary mode is in effect; KERMIT, making sure that binary mode is in effect; or dd.

To see what is in the bundle, enter this command:

```
C:> unbundle /toc xcode.bun
```

The unbundle program lists a short table of contents. To see a full table of contents, enter this command:

```
C:> unbundle /fulltoc xcode.bun
```

To unbundle everything in the bundle and put the files in the current directory, enter this command:

```
C:> unbundle /all xcode.bun
```

To unbundle selected files (`this.x` and `that.x`) and put them in specified directories, enter this command:

```
C:> unbundle xcode.bun /select
this.x   usr2:[xcode]this.x
that.x   usr1:[xcode]that.x
^Z
```

You can also put the selections into a file:

```
C:> create xcode.sel
this.x   usr2:[xcode]this.x
that.x   usr1:[xcode]that.x
^Z
C:> unbundle xcode.bun /select xcode.sel
```

Once you have unbundled the files, they can be executed or compiled in Progress.

Text files can also be bundled. You can specify -ascii on the command line, thereby making ascii the default file type; or you can specify ascii after the filename in the selection list:

```
C:> dir *.p | bundle -ascii pcode.bun
C:> bundle pcode.bun -select
foo.p ascii
^D
C:>
```

Binary and ascii can be mixed in a bundle:

```
C:> bundle allcode.bun -select
foo.p ascii
foo.x binary
^D
C:>
```

Since bundle always appends files to the end, multiple files with the same name might appear in a bundle:

```
C:> bundle allcode.bun -select
foo.p ascii
xcode/foo.p binary
^D
C:>
```

In this example, the filename `foo.p` appears twice and represents two different files. When unbundling this (with -all) `foo.p` (ascii) is unbundled first and then `foo.p` (binary) is unbundled overwriting the other `foo.p`. To unbundle both files, use a selection list to give each file a unique name or put each in a different directory:

```
C:> unbundle allcode.bun -select
foo.p src/foo.p   # This is the ascii foo.p
foo.p xcode/foo.p # This is the binary foo.p
^D
C:>
```

You can also use a selection list to skip over a file by directing it to the null device, as shown in the following UNIX example:

```
$ unbundle allcode.bun -select
foo.p /dev/null
foo.p
^D
$
```

To help differentiate files with identical names, unbundle -full shows files' input file specifications, modify dates, and sizes in bytes; the current directory when the bundle was created; and a time stamp showing when the files were bundled:

```
$ unbundle mystuff.bun -full
# V0 Time-stamp: Thu Apr 15 19:59:19 1993
# V0 Current-directory: C:\TEMP
 mon03.txt      #ascii
# V0 Input-file: MON03.TXT
# V0 File-modify-time: Thu Apr 15 19:57:12 1993
# V0 File-size: 2503
 mon03.txt      #ascii
# V0 Input-file: ..\WVM\DOC\MON03.TXT
# V0 File-modify-time: Sun Apr 04 18:17:48 1993
# V0 File-size: 2655
# V0 Time-stamp: Thu Apr 15 20:00:23 1993
# V0 Current-directory: C:\TEMP
 mon03.txt      #ascii
# V0 Input-file: mon03.txt
# V0 File-modify-time: Thu Apr 15 20:00:10 1993
# V0 File-size: 2666
$
```

(The V0 denotes version 0 of the bundle record format. It allows backward compatibility should this format have to be changed.) You can use -select with -full to see information on selected file names.

NOTE: Binary dump and loads are dramatically faster. See the section on PROUTIL in the *Progress Database Administration Guide and Reference* for more information on binary dump and load.

5.3 Dumping and Loading

You have to supply a facility to dump and reload the application database. Your users need a dump and reload facility to:

- Convert data to ASCII format for transporting among machines or database products
- Increase index efficiency
- Convert from one version of Progress to another
- Upgrade your application if you deploy using time-stamp-based r-code

The Progress 4GL Development System, ProVISION, ProVISION Plus, and WebSpeed products all provide full dump/load capabilities via either the GUI Data Administration Tool, Character Data Dictionary Tool, or command line PROUTIL utility. For more information about these capabilities, see the *Progress Database Administration Guide and Reference*. Other Progress products provide less complete dump/load capabilities. Other Progress products such as Query Results provide less complete dump/load capabilities.

NOTE: Progress Software recommends you use binary dump and load instead of MKDUMP when moving table contents. Refer to the *Progress Database Administration Guide and Reference* for more information about binary dump and load, and loading of sequence values.

The Progress Developer's Toolkit includes sample utilities that you can use to create your own dump/load facility. This section covers these samples and explains how to modify them for your own use.

5.3.1 MKDUMP Utility

This sample utility contains a dump/load facility. When you run the MKDUMP utility, it creates two Progress procedures: one that dumps a database and one that reloads a database. For example, you would run MKDUMP to create a dump/reload facility for the sports database, as follows:

SYNTAX

```
mkdump sports
```

Figure 5–1 shows the files produced by running MKDUMP.

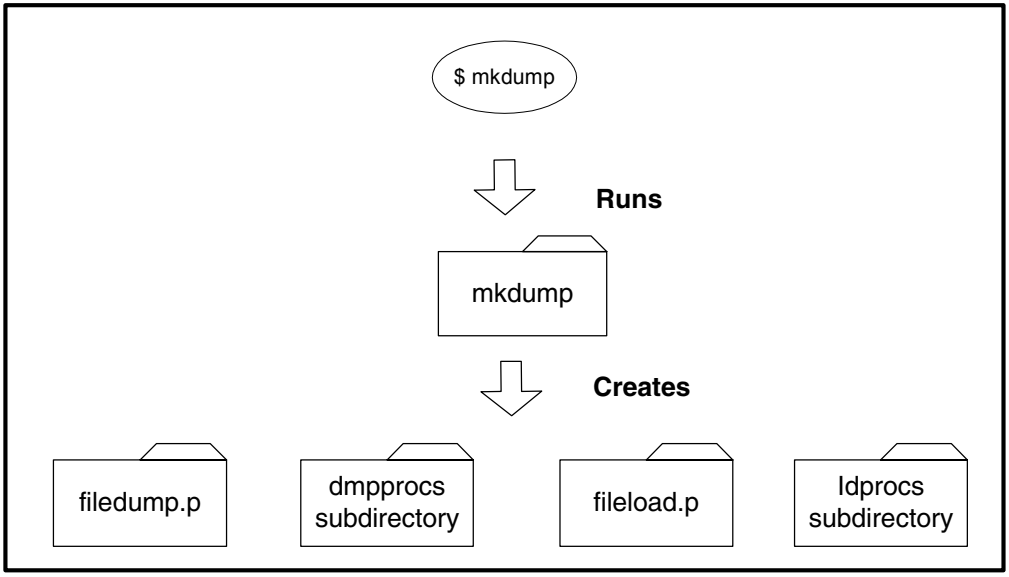


Figure 5–1: Output of mkdump

The MKDUMP utility runs a Progress procedure, mkdump.p, which creates and compiles two other Progress procedures, `filedump.p` and `fileload.p`. The `filedump.p` procedure dumps the files from a database; `fileload.p` loads the files into a database.

In addition, `mkdump.p` creates two directories. The `dmpprocs` subdirectory contains a database dump procedure for each of the files in the database. The `ldprocs` subdirectory contains a database load procedure for each of the files in the database. The `mkdump.p` procedure also precompiles all the procedures it creates. These precompiled procedures may not be appropriate for your use if the deployed database does not have the same name as the original database that MKDUMP was run against.

The MKDUMP utility creates these files and subdirectories in your current working directory. Before distributing them to users, you must move them to your application master directory. You can then supply either the encrypted source or object versions of the `filedump.p` procedure, `fileload.p` procedure, and each of the procedures in the `dmpprocs` and `ldprocs` subdirectories.

You can modify MKDUMP or any of the `mkdump.p`, `filedump.p`, or `fileload.p` procedures as needed. However, once you have created and perhaps modified the dump/load scripts or batch files, be sure to provide a way for users to use these procedures. You should also provide adequate documentation for users to use the facility.

NOTE: The dump/load procedures should be considered part of your application. The decision to distribute r-code or encrypted source versions of your application procedures applies to the dump/load procedures as well.

5.3.2 Using Your Dump/Reload Facility

To dump and reload a database, the user:

- Uses `dumpdb` to dump the current copy of the database
- Uses either `prodb` or `newdb` to make a fresh copy of the basic database
- Uses `reloaddb` to reload the database

Before you supply the `filedump.p, fileload.p,` and subdirectories of dump and reload procedures to a user, you must understand the following points:

- The `filedump.p` and `fileload.p` procedures were written to run from an interactive Progress procedure. If you plan to have users run these procedures from the operating system command level, you must supply an output destination by adding OUTPUT TO *filename* at the top of each procedure. Comments in the `dumpdb` and `reloaddb` files explain these modifications.
- If you modify the procedures, remember to recompile.
- If the database to be loaded does not have the same name as the database that the MKDUMP was run against, then you must recompile the `fileload.p` procedure against the database with the proper name.

5.4 Restricting Database Access

The Developer's Toolkit provides the DBRSTRCT utility to restrict access to a database. If you restrict the database with DBRSTRCT, you cannot compile procedures or upgrade the database at the user site. You must provide a new empty database and a complete set of .r files to upgrade the application, and users must dump and reload their databases.

Because database access restrictions apply to you as well as to your users, you must make a copy of your original, unrestricted and unfrozen database, and store it under a separate name.

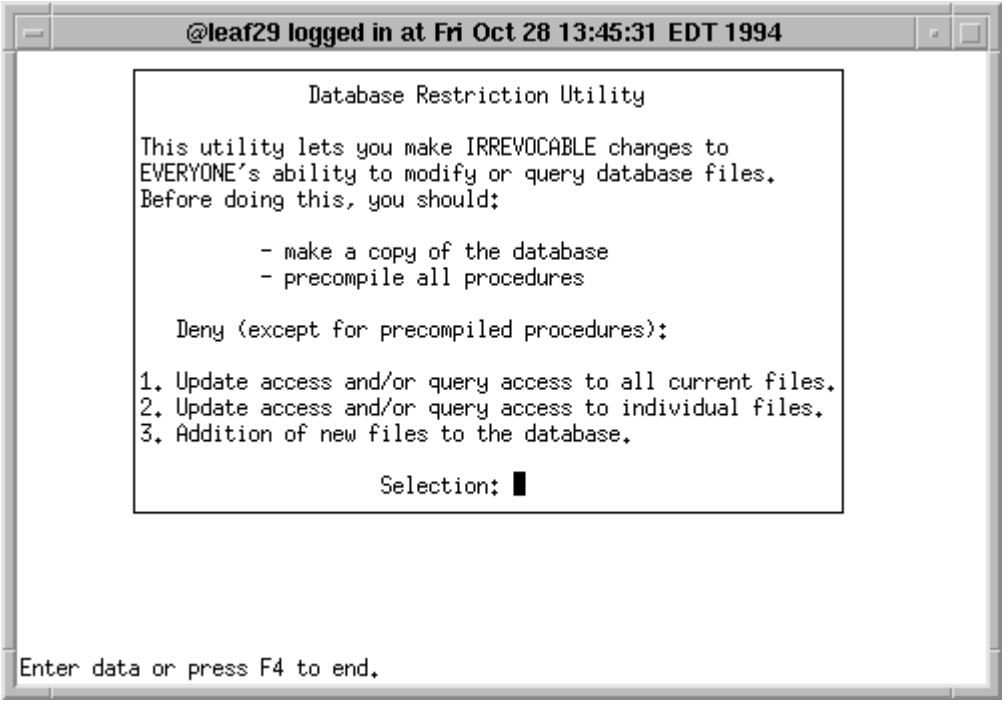
That way, if you want to lessen the restrictions at a later time, you can dump the data in the restricted database and reload it into a copy of the original, unrestricted database.

You use the DBRSTRCT utility to define the kinds of database access all users, including yourself, are allowed. This is the syntax for the DBRSTRCT utility:

SYNTAX (UNIX)

```
dbrstrct database-name
```

In these commands, *database-name* is the name of the database you want to restrict. The utility starts Progress and displays the following menu from which you can choose options that let you define privileges for the update and query activities:



To deny update or query access to all database files that are currently defined, choose 1. A second menu screen appears.

If you choose 1 (Deny updates), at this second menu, users can query but not update the database.

Any procedures you compile against a database before restricting that database are exempt from any restrictions you apply with the DBRSTRCT utility.

Make sure that you compile a complete set of dump procedures before you restrict the database. Otherwise, users will not be able to dump their database.

NOTE: Encrypted source procedures **cannot** be compiled against a restricted database.

A

Toolkit Contents and Reference

This appendix provides reference information for the Developer's Toolkit utilities.

A.1 Toolkit Templates

You can use the templates provided with the Developer's Toolkit, tailor them to the needs of your application, and supply them with your application. The following templates are located in the `DLC/toolkit/samples` directory:

- **runmenu** — Starts single-user Progress session.
- **multmenu** — Starts multi-user Progress session.
- **runbatch** — Starts single-user batch Progress session.
- **multbat** — Starts multi-user batch Progress session.
- **newdb** — Makes a copy of a basic application database.
- **strtsrvr** — Starts a multi-user Progress server.
- **stopsrvr** (supplied on UNIX only) — Stops a multi-user Progress server.
- **deldb** — Deletes a database.
- **dumpdb** — Dumps an application database.
- **reloaddb** — Loads an application database.
- **trimlog** — Discards old information from the database log file.
- **upgrade** — Upgrades an existing application with new data definitions and/or procedures.

A.2 Toolkit Utilities Summary

Table A–1 provides a summary of the different utilities provided with the Developer's Toolkit.

Table A–1: Toolkit Utilities

Utility	Description
dbrstract	Defines the type of access that all users, including you, have to your application.
mkdump	Creates procedures a user runs to dump and reload your database.
xcode	Encrypts source code for shipment to customer sites.
_tlr	Lets you tailor the environment variables used in UNIX scripts.

DBRSTRCT Utility

Defines the access that all users, including you, have to your application database.

SYNTAX

```
dbrstrct database-name
```

PARAMETERS AND QUALIFIERS

database-name

The name of the database for which you want to restrict access.

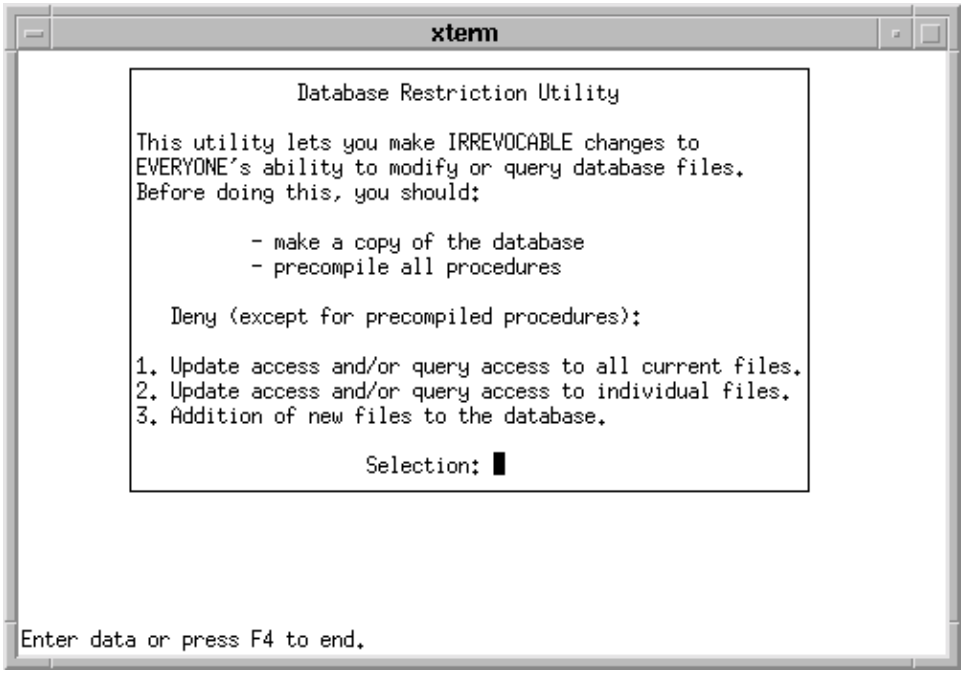
NOTES

- Before you define database access, it is very important that you make a copy of your application database and precompile all of your application procedures against that database. For example, suppose you do not precompile a procedure that updates the database and then you deny update access to the database. You will be unable to compile that procedure against the newly restricted database.
- Keeping a copy of the unrestricted database also lets you redefine the database restrictions at a later time. To redefine database restrictions, dump the data from the restricted database, make a copy of the unrestricted database, define restrictions on that database copy, and reload the data into it.

EXAMPLES

Sample DBSTRCT Command:

```
dbrstrct mydb
```



As you can see from the options listed, you can define access globally, for all files, or you can define access on a file-by-file basis. Any restrictions you define affect all users, including yourself.

If you choose option 1 and deny update access but permit query access, no users, including yourself, will have update access to files or fields in the database, unless that update is being done by a procedure that was precompiled against the database before the database was restricted. However, all users will be able to write procedures that query the database.

MKDUMP Utility

Creates a set of file dump and file load procedures for a database.

SYNTAX

```
mkdump database-name
```

PARAMETERS AND QUALIFIERS

database-name

The name of the database for which you want to create dump and load procedures.

NOTE

- The `filedump.p` and `fileload.p` procedures created by `mkdump` are meant to run from an application menu inside Progress. If you plan to have end users run these procedures directly from the operating system level, you must first modify `filedump` and `fileload`. Comments in the `dumpdb` and `reloaddb` scripts or batch files explain these modifications.

EXAMPLE

Sample MKDUMP Command:

```
mkdump mydb
```

This command creates two main Progress procedures: `filedump.p` and `fileload.p`. In addition, it creates two subdirectories: `dmpprocs` and `ldprocs`. The `dmpprocs` subdirectory contains a single dump procedure for each file in the database; the `ldprocs` subdirectory contains a single load procedure for each file in the database.

An end user runs the `filedump.p` procedure to dump the database. That procedure runs the procedures in the `dmpprocs` subdirectory. To load the database, the user runs `fileload.p`, which runs the procedures in the `ldprocs` subdirectory.

XCODE Utility

Encrypts Progress procedures and include files using either the default or a specified encryption key.

SYNTAX

```
xcode [ -k key ] [ -d directory ] { files | - }
```

PARAMETERS AND QUALIFIERS

key

The encryption key. The *key* is an ASCII character string up to eight characters long. If *key* is not supplied, the default key supplied by Progress is used.

directory

The directory where the encrypted *files* are placed. The specified *directory* must be different from the source directory, because each encrypted file retains the name of its original source file. Existing files in *directory* are overwritten only if they are encrypted.

files

The relative pathnames of files to be encrypted. The pathnames supplied are appended to *directory*. Therefore, you should build a directory structure exactly parallel to the source directory structure, move to the source root, and specify relative pathnames.

- SYS$INPUT

Indicates that source filenames are to be read from standard input.

EXAMPLES

```
cd /applr001
xcode -k key1 -d applx001 *.p
```

```
find applf002/*.p | xcode -k key2 -d applx002 -
```

NOTES

- You must create the full directory structure desired for your encrypted files before running XCODE; XCODE does not create any directories implied by *directory* or *files*.
- The encryption algorithm is based on known techniques; therefore, the code is breakable. If your source code is highly sensitive, consider additional security methods.
- Encrypted include files must use the same *key* as any procedure that includes them.
- Decrypt encrypted procedures with the XCODE option on the COMPILE statement. (The XCODE option is not required if the procedures were encrypted with the default key.) The XCODE option requires the same *key* used with the XCODE utility. Decryption is incremental during compilation. Having the decryption key does not enable you to examine a decrypted version of the source code. The *Progress Language Reference* describes the COMPILE statement in detail.

_tlr Module

Changes the default values of environment variables as defined in various scripts.

SYNTAX

```
_tlr var-name replace-value file-name ...
```

```
_tlr var-name replace-value -< file-list
```

The first syntax format processes one file at a time. The second processes a list of files read from a file.

PARAMETERS AND QUALIFIERS

var-name

A constant character string that identifies the environment variable whose value you want to modify.

replace-value

A constant character string that identifies the new default value of the environment variable.

file-name

The name of the file in which you want to modify the default value of the environment variable.

file-list

The name of a file containing a list of filenames in which you want to modify the value of the environment variable.

EXAMPLE

Several of the scripts provided with Progress and the Toolkit name the DLC environment variable as follows:

```
DLC=${DLC-/usr/dlc}
```

In this example, if the value of the DLC environment variable has been defined, the script uses the value of that variable. Otherwise, it uses `/usr/dlc` as the value of DLC.

To change the default value of the DLC variable to `/tmp/dlc` in a script named `test.dat`, use this command:

```
_tlr DLC /tmp/dlc test.dat
```

This command changes the environment variable line in `test.dat` from:

```
DLC=${DLC-/usr/dlc}
```

to:

```
DLC=${DLC-/tmp/dlc}
```

Index

A

Application databases
  modifying 4–5–4–10

Audience ix

B

Bold typeface
  as typographical convention x

BUNDLE utility 1–2, 5–4–5–11

C

CRC-based r-code
  upgrading applications 4–4

D

Databases
  restricting access 5–13

DataServers 4–4

DBRSTRCT utility 1–2, 2–8, 3–9, 3–13, 5–13–5–15, A–4–A–5

deldb template A–2

Deploying applications
  steps 3–1–3–16

dumpdb template A–2

Dumping 5–11–5–13

E

Encrypted source code 2–4–2–6
  application databases 4–5
  steps for deployment 3–5
  upgrading applications 4–4

Encrypting procedures 5–2–5–4

Encryption keys 2–6

Error messages
  displaying descriptions xvii

Example procedures xv

H

Help
  Progress messages xvii

I

Index modifications 2–6

Internationalization 2–4, 2–8

Italic typeface
  as typographical convention x

K

KERMIT 3–3

Keystrokes x

L

Loading 5–11–5–13

M

Manual
  organization of ix
  syntax notation xi

Messages
  displaying descriptions xvii

MKDUMP utility 1–2, 5–11, A–6

Monospaced typeface
  as typographical convention x

multbat template A–2

multmenu template A–2

N

newdb template A–2

P

Portability
  encrypted source 2–5
  r-code 2–7, 2–9
  unencrypted source 2–3

Procedures
  examples of xv

Product upgrades 4–11

R

R-code 2–6–2–9
  application databases 4–8–4–11
  steps for deployment (CRC) 3–9–3–13
  time-stamp 3–13–3–16

reloaddb template A–2

runbatch template A–2

runmenu template A–2

S

stopsrvr template A–2

Storage media
  space used 2–3

strtsrvr template A–2

Syntax notation xi

T

Templates A–2

Time-stamp-based r-code
  upgrading applications 4–4

_tlr module A–9

trimlog template A–2

Typographical conventions x

U

Unencrypted source 2–2–2–4
  steps for deployment 3–3–3–5
  upgrading applications 4–4

Upgrade templates 4–7, A–2

Upgrades
  Progress products 4–11

Utilities
  BUNDLE. *See* BUNDLE utility
  DBRSTRCT. *See* DBRSTRCT utility
  MKDUMP. *See* MKDUMP utility
  XCODE. *See* XCODE utility

X

XCODE utility 1–2, 5–2–5–4, A–7