

Progress
Internationalization Guide

© 2002 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, Powered by Progress, Progress Fast Track, Progress Profiles, Partners in Progress, Partners en Progress, Progress en Partners, Progress in Progress, P.I.P., Progress Results, ProVision, ProCare, ProtoSpeed, SmartBeans, SpeedScript, and WebSpeed are registered trademarks of Progress Software Corporation in the U.S. and other countries. A Data Center of Your Very Own, Allegrix, Apptivity, AppsAlive, AppServer, ASPen, ASP-in-a-Box, Empowerment Center, Fathom, Future Proof, IntelliStream, OpenEdge, PeerDirect, POSSE, POSSENET, Progress Dynamics, Progress Software Developers Network, SectorAlliance, SmartObjects and WebClient are trademarks or service marks of Progress Software Corporation in the U.S. and other countries.

SonicMQ is a registered trademark of Sonic Software Corporation in the U.S. and other countries.

Vermont Views is a registered trademark of Vermont Creative Software in the U.S. and other countries.

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

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

Progress Software Corporation acknowledges the use of Raster Imaging Technology copyrighted by Snowbound Software 1993-1997, the IBM XML Parser for Java Edition, and software developed by the Apache Software Foundation (http://www.apache.org/).

© 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.

The implementation of the MD5 Message-Digest Algorithm used by Progress Software Corporation in certain of its products is derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm.

May 2002



Product Code: 4524
Item Number: 89431;V9.1D

Contents

Preface . . . . . **xiii**
Purpose xiii
Audience xiii
Organization Of This Manual xiii
Typographical Conventions xv
Syntax Notation xvi
Example Procedures xx
Progress Messages xxiii
Other Useful Documentation xxv
Getting Started xxv
Development Tools xxvi
Reporting Tools xxviii
4GL xxviii
Database xxix
DataServers xxx
SQL-89/Open Access xxx
SQL-92 xxxi
Deployment xxxii
WebSpeed xxxii
Reference xxxiii
SQL-92 Reference xxxiii

1. Developing Applications For Deployment Worldwide **1–1**
1.1 The Importance Of Internationalization and Localization 1–2
1.2 Strategies For Responding To a Global Market 1–3
1.2.1 Internationalized Applications 1–3
1.2.2 Localized Applications 1–4

1.3 How Progress Supports Internationalization 1–4
1.3.1 Multi-byte Applications 1–5
1.3.2 Progress Support For Unicode 1–5
1.3.3 Character Processing 1–5
1.3.4 Translation Manager System 1–6
1.3.5 Progress 4GL Elements 1–6
1.3.6 International Databases 1–6
1.3.7 Progress Messages 1–7
1.3.8 Regional Parameter Files 1–7
1.4 How Progress Products Support Internationalization 1–8
1.5 Design Guidelines 1–9

2. Understanding Code Pages 2–1
2.1 Code Pages and Character Sets 2–2
2.1.1 Code Pages 2–2
2.1.2 Character Sets 2–5
2.2 Environments With Multiple Code Pages 2–7
2.3 Code-page Conversion 2–7
2.3.1 Streams and Code-page Conversion 2–9
2.3.2 Valid and Invalid Code-page Conversions 2–10
2.3.3 Sockets and Code-page Conversion 2–10
2.4 The Undefined Code Page 2–11
2.5 Figuring Out the Code Page an Application Component Uses 2–12
2.5.1 7-bit Character Data 2–12
2.5.2 8-bit Character Data 2–12
2.5.3 Progress Databases 2–13
2.5.4 Character Terminals 2–14
2.5.5 Windows Screen and Keyboard 2–15
2.5.6 Printers 2–16
2.5.7 Table Dump (.d) Files and Other External Text Files 2–17
2.5.8 Progress Libraries 2–17

3. Understanding Character Processing Tables 3–1
3.1 The convmap.dat File and Its Tables 3–2
3.1.1 Character Attribute Tables 3–5
3.1.2 Case Tables 3–6
3.1.3 Collation Tables 3–9
3.1.4 Code-page Conversion Tables 3–11
3.1.5 Modifying convmap.dat Or a File It Includes 3–13
3.1.6 Compiling the convmap.dat File 3–14
3.1.7 Providing Access To the convmap.cp File 3–14
3.1.8 Modifying Collation Tables 3–15
3.1.9 Finding Additional Information On Character Processing Tables Other Than Word-break Tables 3–20

3.2 Word-break Tables 3–21
3.2.1 Why Progress Uses Word-break Tables 3–21
3.2.2 Creating and Modifying Word-break Tables 3–22
3.2.3 Compiling Word-break Tables 3–27
3.2.4 Providing Access To Word-break Tables. 3–27
3.2.5 Associating Word-break Tables With Databases 3–28
3.2.6 Rebuilding the Indexes 3–29

4. Preparing the Code 4–1
4.1 Guidelines and Methodology 4–2
4.1.1 Structuring Source Code 4–2
4.1.2 Processing By Characters 4–4
4.1.3 Using Variables 4–5
4.1.4 Coding With Translation In Mind 4–5
4.2 Input and Output 4–6
4.2.1 Keyboards 4–6
4.2.2 Printers 4–7
4.3 Data-processing Issues 4–8
4.3.1 Numeric Formats 4–8
4.3.2 Measurements 4–9
4.3.3 Currencies 4–10
4.3.4 Dates 4–11
4.3.5 Time 4–13
4.3.6 Addresses 4–13
4.4 Sorting Data 4–14
4.4.1 Local Conventions 4–14
4.4.2 Using Collation Tables 4–15
4.4.3 Using Case Tables. 4–16
4.5 Compiling Translated Applications 4–18

5. Preparing the User Interface 5–1
5.1 Screen Layout and Composition 5–2
5.1.1 Equipment Differences. 5–2
5.1.2 Culturally-specific Issues 5–4
5.2 Graphics and Icons 5–11
5.2.1 Text In Graphics and Icons 5–11
5.2.2 Images To Avoid In Graphics and Icons 5–11

5.3 Designing With the AppBuilder 5–13
5.3.1 Localizing SmartObject Masters 5–14
5.3.2 Localizing SmartObject Instances 5–15
5.3.3 Using Multiple Layouts 5–18
5.4 Designing To Allow For Translation 5–19
5.4.1 Text Expansion and Contraction 5–19
5.4.2 Message Text 5–21
5.4.3 Layout 5–21

6. Using Databases 6–1
6.1 Progress Database Name Restrictions 6–2
6.2 Empty Databases 6–2
6.2.1 Empty Databases Supplied By Progress 6–2
6.2.2 Creating an Empty Database In a Particular Code Page 6–3
6.3 Scanning Databases For Character Conflicts 6–5
6.3.1 New Characters 6–5
6.3.2 Using PROUTIL To Perform the Scan 6–8
6.4 Converting a Database and Its Data To a Different Code Page 6–10
6.5 Loading Table Dump Files 6–11
6.5.1 Code-page Trailers 6–11
6.5.2 Techniques For Loading Table Dump Files 6–12
6.6 Specifying -cpinternal and -cpstream With Database Utilities 6–13

7. Using SQL-92 7–1
7.1 Starting Database Servers 7–2
7.2 Using SQL-92 Database Servers 7–3
7.3 Code-page Conversion In SQL-92 Applications 7–4
7.4 Using SQL-92 Clients 7–7
7.4.1 ESQL-92 Clients 7–7
7.4.2 Java/JDBC Clients 7–7
7.4.3 ODBC Clients 7–7
7.4.4 Characteristics Of SQL-92 Clients 7–8
7.5 Using SQL-92 Utilities 7–9
7.5.1 SQLDUMP 7–9
7.5.2 SQLLOAD 7–10
7.5.3 SQLSCHEMA 7–12
7.6 Using the SQL-92 Language 7–12
7.6.1 The Unit Of Length When Working With Character Strings . . . 7–13
7.6.2 The Maximum Number Of Bytes Required By Fields Of Type CHAR and VARCHAR 7–13
7.6.3 SQL-92 Elements That Support Internationalization and Localization 7–14
7.6.4 Format Specifiers Allowed With the TO_CHAR and TO_DATE Functions 7–25

8. Using Multi-byte Code Pages 8–1
8.1 Definitions For Key Terms 8–2
8.1.1 Terms For Code Pages 8–2
8.1.2 Terms For Characters 8–2
8.1.3 Terms For Bytes 8–3
8.1.4 Illustrations Of the Terms 8–3
8.2 Progress Support 8–4
8.3 Input 8–7
8.3.1 Using the Keyboard and Mouse 8–7
8.3.2 Using the 4GL 8–12
8.3.3 Automatic Input Validation 8–14
8.4 Output 8–15
8.4.1 The FORMAT Phrase 8–15
8.4.2 Printing 8–18
8.4.3 Character-client Color Limit 8–19
8.5 Inside the Multi-byte Application 8–20
8.5.1 Distinguishing Characters, Bytes, and Columns 8–20
8.5.2 Techniques For Working With Multi-byte Characters 8–21
8.5.3 4GL Support For Processing Multi-byte Characters 8–28
8.6 Issues Specific To Multi-byte Code Pages 8–30
8.6.1 Progress Support For Multi-byte Code Pages 8–31
8.6.2 Valid and Invalid Code-page Conversions 8–32
8.6.3 User-defined Characters 8–33
8.6.4 Collating Multi-byte Characters 8–34
8.6.5 Default Word-break Behavior Of Characters In Multi-byte Code Pages 8–36
8.7 Guidelines For Using Multi-byte Characters 8–37

9. Using Unicode 9–1
9.1 What Is Unicode 9–2
9.2 Why Use Unicode 9–2
9.2.1 The Limits Of Multiple Code Pages 9–2
9.2.2 The Advantages Of Unicode 9–3
9.3 Using Unicode With Progress Products 9–4
9.4 . . . . . Using Unicode With Progress Databases 9–5
9.4.1 Converting a Progress Database To UTF-8 Using the PROUTIL CONVCHAR Utility 9–5
9.4.2 Converting a Progress Database To UTF-8 Using Dump and Load 9–6
9.4.3 Compiling, Storing, and Applying the UTF-8 Word-break Rules To a Database 9–7
9.5 Using Unicode With Progress Applications 9–8
9.6 Guidelines For Using Unicode 9–10

10. Deployment and Configuration 10–1
10.1 The progress.ini File and the Windows Registry 10–2
10.1.1 [Startup]. 10–2
10.1.2 [fonts] 10–3
10.1.3 Specifying Scripts 10–4
10.2 The PROTERMCAP File 10–5
10.3 The PROMSGS File 10–5
10.4 Parameter Files 10–6
10.5 The conmgr.properties File 10–8
10.6 Report Builder Parameters 10–9
10.7 Additional Guidelines 10–10
10.7.1 Running Applications Translated Using the Translation Manager 10–10
10.7.2 Using the UNIX Character Client With Double-byte Code Pages 10–10
10.7.3 Specifying -cpinternal and -cpstream With Database Utilities. 10–10

A. Progress Resources A–1
A.1 Progress Product Support For Multi-byte Characters A–2
A.2 Files In the DLC/prolang Directory A–5
A.3 Startup Parameters and Settings A–6
A.4 Progress 4GL A–10
A.5 Progress SQL-92 . . . . . A–17
A.6 Utilities A–32
A.7 Determining the Code Page A–33

B. Character Processing Table Formats B–1
B.1 Character Attribute Table B–2
B.2 Case Table B–3
B.3 Collation Table B–4
B.4 Code-page Conversion Table B–6
B.5 Word-break Table B–8

Index Index–1

Figures

Figure 2–1: The IBM850 Code Page 2–2
Figure 2–2: The ISO8859–1 Code Page 2–3
Figure 2–3: The Character Set Of the IBM850 Code Page 2–5
Figure 2–4: The Character Set Of the ISO8859–1 Code Page 2–6
Figure 2–5: A Russian Application Where Each Component Has a Different Code Page 2–7
Figure 2–6: Code-page Conversion 2–8
Figure 3–1: Code Page 1256's Character Attribute Table 3–5
Figure 3–2: Code Page ISO8859–15's BASIC Case Table 3–7
Figure 3–3: Code Page 1253's GREEK Collation Table 3–10
Figure 3–4: Table For Converting From Code Page 1254 To Code Page 709 3–11
Figure 3–5: Sample _tran.df File 3–17
Figure 3–6: The big5-bas.wbt Word-break Table 3–22
Figure 5–1: SmartObject Labels 5–14
Figure 5–2: SmartFolder Attributes Dialog Box 5–15
Figure 5–3: Localized SmartFolder 5–16
Figure 5–4: Graphical Navigation SmartPanel 5–16
Figure 5–5: Navigation SmartPanel Attributes Dialog Box 5–17
Figure 6–1: A Typical Code-page Trailer 6–11
Figure 7–1: Code-page Conversion In an SQL-92 Application 7–5
Figure 7–2: Code-page Conversion In a 4GL Application 7–6
Figure 8–1: Four Characters From a Double-byte Code Page 8–3
Figure 8–2: Three Characters From a Triple-byte Code Page 8–3
Figure 8–3: Application That Does Not Use an Input Method Editor 8–9
Figure 8–4: Application That Uses an Input Method Editor 8–11
Figure 8–5: A Single-byte Character Overlaying a Lead Byte 8–22
Figure 8–6: Result Of a Single-byte Character Overlaying a Lead Byte 8–22
Figure 8–7: String Produced By an OVERLAY Statement Whose Unit of Measure Is the Character 8–23
Figure 8–8: Sorting Double-byte Characters 8–35
Figure 10–1: [Startup] Section of the progress.ini File 10–2
Figure 10–2: [fonts] Section of the progress.ini File 10–3
Figure B–1: Format Of the Character Attribute Table B–2
Figure B–2: Format Of the Case Table B–3
Figure B–3: Format Of the Collation Table B–4
Figure B–4: Format Of the Code-page Conversion Table B–6
Figure B–5: Portion Of a Code-page Conversion Table B–7

Tables

Table 3–1: Contents Of arabic.dat 3–2
Table 3–2: Converting to UTF–8 3–12
Table 3–3: Converting From UTF–8 3–12
Table 3–4: Word-delimiter Attributes 3–23
Table 4–1: International Numeric Formats 4–8
Table 4–2: RCODE–INFO Handle Attributes 4–19
Table 5–1: Date and Time Formats 5–6
Table 5–2: Currency Formats 5–8
Table 5–3: The Difference In Text Length After Translation 5–20
Table 6–1: New Characters In 1252 and ISO8859–15 6–5
Table 6–2: Characters To Check For Before Converting To MS1252 6–6
Table 6–3: Characters To Check For Before Converting To ISO8859–15 6–7
Table 7–1: Localizable Startup Parameters and Settings 7–2
Table 7–2: SQL-92 Language Elements That Support Internationalization and Localization 7–14
Table 7–3: Format Specifiers Allowed With TO_CHAR() and TO_DATE() 7–25
Table 8–1: Support For Multi-byte Characters 8–4
Table 8–2: 4GL Elements That Simulate Key Presses and Mouse Clicks 8–13
Table 8–3: Automatic Validation Of Multi-byte Input 8–14
Table 8–4: Byte Count and Column Count 8–20
Table 8–5: Lead Byte and Trail Byte Values 8–26
Table 8–6: 4GL Elements For Processing Multi-byte Applications 8–28
Table 8–7: Multi-byte Code Pages That Progress Supports 8–31
Table 8–8: Valid Double-byte To Double-byte Code-page Conversions 8–32
Table 8–9: Japanese Collation Order By Character Type 8–34
Table 8–10: Default Word-break Behavior Of Characters In Multi-byte Code Pages 8–36
Table 9–1: Unicode’s Advantages 9–3
Table 10–1: Localizable Settings In the [Startup] Section Of the progress.ini File 10–3
Table 10–2: Localizable Startup Parameters 10–7
Table 10–3: Localizable Properties In the conmgr.properties File 10–8
Table 10–4: Localizable Report Builder Parameters 10–9
Table A–1: Support For Multi-byte Characters A–2
Table A–2: Files In DLC/prolang A–5
Table A–3: Startup Parameters A–6
Table A–4: Localizable Progress Explorer Properties A–9
Table A–5: 4GL Elements That Support Internationalization and Localization A–10
Table A–6: 4GL Elements That Support Multi-byte Characters A–13
Table A–7: SQL-92 Language Elements That Support Internationalization and Localization A–17

Table A–8: Format Specifiers Allowed With TO_CHAR() and TO_DATE() A–28
Table A–9: Utilities Used For Internationalization and Localization A–32
Table A–10: Determining the Code Page A–33
Table B–1: Word-delimiter Attributes B–10

Procedures

p-snhndl.p 4–12

Preface

Purpose

This book is a guide to developing Progress applications for a worldwide market. It describes important issues that a developer should consider when designing, creating, and modifying code that works in many cultures across the globe, or when translating and customizing Progress products for international deployment.

Audience

This book is intended for programmers who want to develop Progress applications in an international context. It does not assume Progress experience.

Organization Of This Manual

Chapter 1, “Developing Applications For Deployment Worldwide”

Describes an approach to designing international applications. Outlines how Progress supports international development efforts. Defines terms related to internationalization and localization.

Chapter 2, “Understanding Code Pages”

Provides a theoretical and practical introduction to code pages, character sets, and code-page conversion.

Chapter 3, “Understanding Character Processing Tables”

Provides a theoretical and practical introduction to character attribute tables, case tables, collation tables, code-page conversion tables, and word-break tables.

Chapter 4, “Preparing the Code”

Suggests programming conventions and specific techniques for creating the back end of an international or localized application.

Chapter 5, “Preparing the User Interface”

Presents the cultural and linguistic issues that affect user-interface design and suggests techniques for using the Progress AppBuilder to manage localizing interface objects.

Chapter 6, “Using Databases”

Tells you how to use databases in applications to be deployed across multiple locales.

Chapter 7, “Using SQL-92”

Tells you how to use SQL-92 in applications to be deployed across multiple locales.

Chapter 8, “Using Multi-byte Code Pages”

Tells you how to use multi-byte code pages in applications to be deployed across multiple locales. Defines terms related to multi-byte code pages.

Chapter 9, “Using Unicode”

Provides information on Progress support for multilingual applications through implementation of the Unicode Standard character set.

Chapter 10, “Deployment and Configuration”

Provides guidelines for deploying and configuring international applications.

Appendix A, “Progress Resources”

Lists the Progress resources that support international development. Describes resources at the system level, the 4GL level, the SQL-92 level, and the utility level.

Appendix B, “Character Processing Table Formats”

Presents formats for the character attribute table, the case table, the collation table, the code-page conversion table, and the word-break table.

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 } )
  ```

  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 Tutorials*, 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

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.

Creating a Listing Of the Procedure Libraries

Creating a listing of the source files from a procedure library involves running PROENV to set up your Progress environment, 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 text file `prodoc.txt` which contains the file listing for the prodoc.pl library.

```
PROLIB %DLC%\src\prodoc.pl -list > prodoc.txt
```

Extracting Source Files From Procedure Libraries On Windows Platforms

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 bin directory under the DLC environment variable to your PATH environment variable (PATH=$DLC/bin:$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/proghand.

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)
```

On the Windows platform, use Progress online help to get more information about Progress messages. Many Progress tools include the following Help menu options to provide information about messages:

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

On the UNIX platform, 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 bulletin that provides a list of new and changed features by release, a list and description of changes to documentation by release, and critical information about product changes that might require changes to existing code and configurations.

This bulletin also provides information about where to go for detailed information about the new and changed features and documentation.

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 and shutdown commands that you use at the command line, and the startup parameters that you use for Progress processes. This guide also provides information about parameter usage and parameter files.

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 the external programming interfaces supported by Progress. This manual covers the Host Language Call (HLC) Interface, the system clipboard, named pipes, shared libraries and DLLS, Windows Dynamic Data Exchange (DDE), COM objects, ActiveX Automation, ActiveX controls, sockets, XML, SAX, and the SonicMQ 4GL Adapter.

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 on the Web

A manual that describes how to use the new WebClient, AppServer Internet Adapter, SmartObjects, and SonicMQ Adapter to create applications tailored for Internet, intranet, and extranet environments.

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 as follows:

- *Progress/400 Product Guide*
- *Progress DataServer for Microsoft SQL Server Guide*
- *Progress DataServer for ODBC Guide*
- *Progress DataServer for ORACLE 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 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/products/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, build, and deploy Java and ActiveX applications, and Java applets that run as clients of the Progress AppServer. This guide includes information about how to expose the AppServer as a set of Java classes or as an ActiveX server, and how to choose an Open Client distribution package for run time.

Progress SQL-89 Guide and Reference

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

SQL-92

Progress Embedded SQL-92 Guide and Reference

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

Progress JDBC Driver Guide

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

Progress ODBC Driver Guide

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

Progress SQL-92 Guide and Reference

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

Deployment

Progress Client Deployment Guide

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

Progress Developer's Toolkit

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

Progress Portability Guide

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

WebSpeed

Getting Started with WebSpeed

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

WebSpeed Installation and Configuration Guide

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

WebSpeed Developer's Guide

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

WebSpeed Product Update Bulletin

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

Welcome to WebSpeed (Hard copy only)

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

Reference

Pocket Progress (Hard copy only)

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

Pocket WebSpeed (Hard copy only)

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

Writers: The following section belongs in the SQL-92 books only.

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.

Understand the New SQL: A Complete Guide

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

1

Developing Applications For Deployment Worldwide

This chapter summarizes the issues connected with developing Progress applications for deployment worldwide. The chapter includes the following sections:

- The Importance Of Internationalization and Localization
- Strategies For Responding To a Global Market
- How Progress Supports Internationalization
- How Progress Products Support Internationalization
- Design Guidelines

1.1 The Importance Of Internationalization and Localization

Imagine that you have just bought an imported video cassette recorder (VCR). You open the box, find an instruction booklet written in your language, and begin reading. You soon notice that:

- The language of the booklet sounds stilted.
- The punctuation and capitalization do not use the rules of your language.
- In the section on programming the VCR to record in your absence, dates and times are not formatted the way you are used to
- Some of the instructions show hand gestures that have no meaning in your country.
- Some of the illustrations contain text that is untranslated. You wonder what this text means.

You conclude that the VCR was not designed with you in mind, that it might not perform satisfactorily for you, and that it might not be of good quality. You might decide to exchange it for a competing model.

The same scenario can occur with software. Consider a product that you market internationally and that you have not fully localized. Your international customers, noticing that the product was not designed with them in mind, might conclude that the product will not perform satisfactorily for them, and might exchange it for a product from one of your competitors.

To avoid this, make sure that any application you market internationally is adapted appropriately for each region’s users.

1.2 Strategies For Responding To a Global Market

In today's business climate, managing information across an enterprise has come to mean managing information across linguistic and cultural regions. The barriers that information systems have to overcome are no longer just variations in business applications from department to department—sharing data between sales and distribution, for example. Now you have the additional requirement of sharing data across a border with a neighboring country, with a corporation on the opposite side of the globe, or with an international corporation's offices located in the world's major cities.

Even if your development plans do not call for sharing data internationally—perhaps you are developing a stand-alone application to sell worldwide—your application must "translate" well into the many business and cultural contexts of the global marketplace.

There are several strategies that software developers can use to address the demands of worldwide deployment. The development of internationalized and localized applications has improved due to increased use of applications worldwide. The advantages and disadvantages are known. The common denominator for success is that these strategies involve thorough research and careful planning.

1.2.1 Internationalized Applications

An *internationalized application* is one that has been designed and implemented so that it can be modified to adapt to all cultural or linguistic contexts. Thus, internationalizing an application is a prerequisite to creating specific language editions that take regional business conventions into consideration.

Although an internationalized application is often incomplete until it is adapted to a specific region, you can consider an internationalized application deployable if, for example, you supply customizable modules that developers or system administrators on site adapt to their locale. Or if you allow users to tailor preferences, such as the language of the interface, the format of reports, measurement systems, and calendar and date formats, you can consider your application to be internationalized.

The major design principle for an internationalized application is to allow regional aspects to be changeable without requiring modifications to code. You should not hard code anything that might be affected by regional differences—from labels on the user interface to procedures that back up databases based on date information obtained from the operating system that is assumed to be in a specific date format.

1.2.2 Localized Applications

A *locale* is a region distinguished by language, culture, or business practice. A locale need not be in another country; it can be a region characterized by a language, such as the French-speaking part of Switzerland, or a region characterized by different business practices, such as a province or state that has a different tax structure.

A *localized application* is an application you have customized for a specific locale.

From the standpoint of an efficient development process, a localized application builds upon or extends an internationalized application. Working with a base internationalized application that you complete with locale-specific modules allows you to keep the benefits of centrally maintainable code while retaining flexibility to address each locale's cultural, linguistic, and business needs.

This book refers to localized applications as *language editions*, since the type of localization it focuses on is language based. However, localizing an application, or creating a language edition, involves much more than translating the user interface. In addition to the text components of any user interface, a localized application ensures the proper presentation and processing of data that a particular locale requires.

A successfully localized application should appear to the users as though it were developed locally. It is often a challenge that requires considerable research and access to native consultants who have experience in the geographic region and business sector your application is targeting.

1.3 How Progress Supports Internationalization

All components of Progress support your international development efforts, including the Application Development Environment (ADE) tools, the 4GL, and the database. In addition, as part of the installation media, Progress provides files and templates that you can use to set up systems and applications for many countries. The following sections provide an overview of how you can use the different parts of Progress to develop and deploy applications worldwide.

Progress supplies files that support many *character sets.* A character set is a specific collection of alphabetic, numeric, and graphic characters that are related to one another. A character set can also include communication and printer control codes, such as a backspace, tab, and a blank.

1.3.1 Multi-byte Applications

Progress supports multi-byte characters. This means that you can develop applications for countries whose languages require multi-byte character sets.

Double-byte characters comprise a subset of multi-byte characters. Languages that use double-byte characters include Chinese, Japanese, and Korean.

For more information on multi-byte languages, see Chapter 8, "Using Multi-byte Code Pages." For information on running multi-byte character clients, see Chapter 10, "Deployment and Configuration."

1.3.2 Progress Support For Unicode

Progress supports multilingual applications by implementing the Unicode Standard character set, an international standard that intends to take into account all the languages of the world, and that currently includes all languages used in business today. For more information on Unicode, see Chapter 9, "Using Unicode."

1.3.3 Character Processing

Character processing is a critical function for exchanging information across language barriers. A character set is the set of all the symbols used by a language to represent it in writing. These symbols include letters, numbers, punctuation, and other communication symbols. Character sets must be converted into numeric code the computer understands. Code pages map characters to numeric codes. Code pages can vary depending on the character set of the language.

Character processing involves managing and working with code pages that represent character sets. Progress supports a wide range of character sets and code pages that accommodate many languages. The Progress database and application environment have administrative options that allow you to manage code pages and control where conversions take place. See Chapter 2, "Understanding Code Pages," and Chapter 3, "Understanding Character Processing Tables," for a discussion of the issues and for instructions on how to manage code pages for your databases and applications.

1.3.4 Translation Manager System

The Translation Manager System—the Translation Manager and the Visual Translator—supports the translation of Progress applications, including the graphical or character interface and application code. These tools offer a visual image of how the user interface will appear. The semantic and spatial context allows the translator to better resolve usage questions when translating text phrases.

A project manager responsible for localization of an application can use the Translation Manager to maintain glossaries and create a translation kit that packages an application's text elements, such as user-interface labels and messages. The tool places all translatable strings into a translation database. It also provides filters to limit the set of strings to just those the translator requires.

The Visual Translator is the tool for the translator. It allows the translator to visualize text strings in the context of the user interface. The translator can make optimal choices and resolve text-expansion issues immediately. For more information, see the *Progress Translation Manager Guide* and the *Progress Visual Translator Guide*.

1.3.5 Progress 4GL Elements

Progress lets you adjust conventions for dates and numeric formats through startup parameters, such as Date Format (-d) and European Numeric Format (-E), and through session handles. The character- and string-processing elements of Progress fully support multi-byte character sets. See Appendix A, "Progress Resources," for a list of Progress elements that support internationalization.

1.3.6 International Databases

As part of the installation media, Progress supplies empty databases that support the language and collation standards of many languages. The databases are located in the `DLC/prolang` subdirectories. Look for the subdirectory name that matches a particular language. For example, the empty database that you might use to build a Russian application is `DLC/prolang/rus/empty.db`.

These empty databases provide a database labelled with the appropriate code page and collation table for a language. However, if you are developing applications for a language or region that is not represented in `DLC/prolang`, Progress has a utility, PROUTIL, that you can use to set up a unique database. For more information on PROUTIL, see the *Progress Database Administration Guide and Reference*.

1.3.7 Progress Messages

The text used in Progress messages is contained in the promsgs file. Progress provides various language editions of the promsgs file in the DLC/prolang subdirectories. Each file has an extension that identifies its language. The following excerpt of the Dutch version of Progress messages is from DLC/prolang/dut/promsgs.dut :

```
>0062ISO8859-1
%gbfblst -- Te veel block nivo's. Verhoog -nb optie. (1)%GSYSTEM ERROR:
bfblnd -- geen ar aktiref (2)%GSYSTEM ERROR: bfreg -- geen tak.
```

To run Progress with a certain version of promsgs, set the PROMSGS environment variable to the appropriate file location. By default, Progress retrieves messages from DLC/promsgs.You can also change the PROMSGS language dynamically, from within your application.

1.3.8 Regional Parameter Files

A useful technique for controlling a Progress client session or server is to use a parameter file (.pf file) with a startup or connection command. Progress provides parameter files that set up Progress sessions appropriately for a wide range of countries. You can use parameter files to specify the correct code-page settings for international applications. For example, you could use the following parameter file to specify the code page settings for a Swedish application:

```
-d ymd -lng "Swedish" -cpcoll Swedish -cpcase Basic
```

The Progress session you start with this parameter file displays dates by year, month, and day. It runs the Swedish language edition of the code that was translated using the Progress Translation Management System (-lng “Swedish”). Progress sorts character data according to Swedish alphabetization rules (-cpcoll Swedish). And it follows the guidelines in the Basic case table when converting between uppercase and lowercase characters (-cpcase Basic).

The international parameter files are located in the DLC/prolang subdirectories. Parameter files are region or country specific rather than language specific because parameter files set options that can vary from country to country. The DLC/prolang/ger directory has parameter files for Austria, Germany, and Switzerland to account for the differences among these German-speaking countries.

1.4 How Progress Products Support Internationalization

In addition to the Translation Manager System, other Progress products support your internationalization efforts.

The Progress Database supports, among other code pages, the Unicode Standard character set, which simplifies multinational deployment. See Chapter 9, "Using Unicode," for information on Unicode. See the *Progress Database Design Guide* for a discussion of designing and creating effective databases.

The Progress AppServer provides support for multiple code pages and languages. The AppServer supports the same code-page options as the 4GL client and allows you to specify a code page that is different from that used by the AppServer client. You can localize your application for a specific language using the same mechanism you use for the 4GL client.

The Progress Open Client Toolkit supports international Java and ActiveX Controller client applications by communication with the Progress AppServer using Unicode. The AppServer then converts between Unicode and the internal code page defined for the AppServer. See Chapter 9, "Using Unicode," for information on Unicode and UTF–8. See *Building Distributed Applications Using the Progress AppServer* for complete information on the AppServer.

The Progress DataServers for ORACLE and ODBC and the Progress/400 DataServer support international applications. Configuring these products for an international locale involves configuring two environments—the Progress environment and the non-Progress data-source environment. For example, you must properly configure ORACLE's National Language Support environment or understand how to specify code pages for other databases. See the *Progress DataServer for ORACLE Guide*, the *Progress DataServer for ODBC Guide*, or the *Progress/400 Product Guide* for information on managing code pages and conversions between the two environments.

WebSpeed supports international applications. See the *WebSpeed Installation and Configuration Guide* for information on designing international internet applications and specific connection requirements.

1.5 Design Guidelines

Before you start developing an international application, you must carefully research the markets that you are targeting so that you can list the technical, cultural, legal, and linguistic aspects that your application must handle. Next, plan in detail how your application will address these issues in the user interface and in back-end processing. Finally, select an application structure and coding conventions that protect your productivity while supporting the modularity that international applications require.

Use the following guidelines to help you assess the requirements of each market:

- Research hardware and software issues that affect the input and output of data.
- Research business standards, legal issues, and local conventions.
- Research cultural conventions that affect the user interface.
- Research language issues that affect the processing and display of data.
- Plan your localization and translation strategies.
- Design a base user interface that can accommodate different languages and cultural conventions.
- Implement a modular structure for your application that isolates language- or region-specific routines or objects.
- Follow coding standards that handle multi-byte characters correctly, even if you do not initially plan to deploy your applications in markets that require multi-byte processing.
- Consider whether your application requires multilingual support and if the Progress implementation of the Unicode Standard meets your requirements.

For more information on design guidelines, see Chapter 5, “Preparing the User Interface.”

For more information on programming and system administration issues that you must consider when developing and deploying an international Progress application, see Chapter 4, “Preparing the Code,” and Chapter 10, “Deployment and Configuration.”

For more information on issues that affect applications that use multi-byte character sets, see Chapter 8, “Using Multi-byte Code Pages,” and Chapter 9, “Using Unicode.”

2

Understanding Code Pages

Suppose you walk up to a computer running the Icelandic edition of a popular operating system, start up a text editor, enter some Icelandic text, and save the result as a text file. To you, the text file consists of a series of letters, numbers, punctuation, control codes, such as the carriage return, and other characters. But to the computer, the text file consists of a series of numeric values, each of which corresponds to a particular character. For the computer to interpret the numeric values correctly, it needs a table that associates each numeric value with the corresponding character. This table is the code page.

Code pages play key roles in applications to be adapted to multiple locales, so this chapter examines them in depth in the following sections:

- Code Pages and Character Sets
- Environments With Multiple Code Pages
- Code-page Conversion
- The Undefined Code Page
- Figuring Out the Code Page an Application Component Uses

2.1 Code Pages and Character Sets

Code pages and character sets are not the same.

2.1.1 Code Pages

A *code page* is a table that assigns a numeric value to each element in a collection of letters, numbers, punctuation, control codes, and other characters. The assignment is one-to-one, meaning that no two characters are assigned the same numeric value and that no two numeric values are assigned the same character.

Figure 2–1 shows the characters and numeric values of the IBM850 code page, widely used in western Europe and the Americas. The white area contains the characters, while the two light gray areas contain the numeric values. By convention, numeric values are in hexadecimal (hex). To compute the numeric value of a character, add the numeric value at the top of the character's column to the numeric value at the far left of the character's row. For example, the numeric value of the character "Ö" is 99 $_{hex}$ (90 $_{hex}$ + 9 $_{hex}$), which equals 153 in decimal.

	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
0		0	@	P	‘	p	Ç	É	á		└	ð	Ó	(SHY)
1	!	1	A	Q	a	q	ü	æ	í		┴	Ð	ß	±
2	“	2	B	R	b	r	é	Æ	ó		┬	Ê	Ô	‗
3	#	3	C	S	c	s	â	ô	ú		├	Ë	Ò	¾
4	$	4	D	T	d	t	ä	ö	ñ	┤	─	È	õ	¶
5	%	5	E	U	e	u	à	ò	Ñ	Á	┼	ı	Õ	§
6	&	6	F	V	f	v	å	û	ª	Â	ã	Í	µ	÷
7	’	7	G	W	g	w	ç	ù	º	À	Ã	Î	þ	¸
8	(	8	H	X	h	x	ê	ÿ	¿	©	╚	Ï	Þ	°
9	)	9	I	Y	i	y	ë	Ö	®	╣	╔	┘	Ú	¨
A	*	:	J	Z	j	z	è	Ü	¬	║	╩	┌	Û	·
B	+	;	K	[	k	{	ï	ø	½	╗	╦	█	Ù	¹
C	,	<	L	\	l	|	î	£	¼	╝	╠	▄	ý	³
D	-	=	M	]	m	}	ì	Ø	¡	¢	═	¦	Ý	²
E	.	>	N	^	n	~	Ä	×	«	¥	╬	Ì	¯	■
F	/	?	O	_	o	⌂	Å	ƒ	»	┐	¤	▀	´	(NBSP)

Figure 2–1: The IBM850 Code Page

Figure 2–2, which uses the same format as Figure 2–1, shows the characters and numeric values of the ISO8859–1 code page, also widely used in western Europe and the Americas.

	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
0		0	@	P	,	p			(RSP)	°	À	Ð	à	ð
1	!	1	A	Q	a	q			¡	±	Á	Ñ	á	ñ
2	"	2	B	R	b	r			¢	²	Â	Ò	â	ò
3	#	3	C	S	c	s			£	³	Ã	Ó	ã	ó
4	$	4	D	T	d	t			¤	´	Ä	Ô	ä	ô
5	%	5	E	U	e	u			¥	µ	Å	Õ	å	õ
6	&	6	F	V	f	v			¦	¶	Æ	Ö	æ	ö
7	'	7	G	W	g	w			§	·	Ç	×	ç	÷
8	(	8	H	X	h	x			¨	¸	È	Ø	è	ø
9	)	9	I	Y	i	y			©	¹	É	Ù	é	ù
A	*	:	J	Z	j	z			ª	º	Ê	Ú	ê	ú
B	+	;	K	[	k	{			«	»	Ë	Û	ë	û
C	,	<	L	\	l	|			¬	¼	Ì	Ü	ì	ü
D	-	=	M	]	m	}			(SHY)	½	Í	Ý	í	ý
E	.	>	N	^	n	~			®	¾	Î	Þ	î	þ
F	/	?	O	_	o				¯	¿	Ï	ß	ï	ÿ

Figure 2–2: The ISO8859–1 Code Page

Here are some other important points about code pages:

- The term *code point* refers to an element of a code page—that is, to a character and its numeric value. For example, in the ISO8859–1 code page, code point 4B contains the character “K” and the value $4B_{hex}$.

- A code page can be single byte, double byte, or triple byte, depending on the maximum size of the numeric value in each entry. An example of a double-byte code page is BIG–5, used for Traditional Chinese. An example of a triple-byte code page is UTF–8, an encoding of Unicode. For more information on using multi-byte code pages in applications, see Chapter 8, “Using Multi-byte Code Pages,” and Chapter 9, “Using Unicode.”

2.1.2 Character Sets

In contrast to a code page, a *character set* is merely a collection of letters, numbers, punctuation, and control codes in no particular order. Figure 2–3 shows the character set of the IBM850 code page.

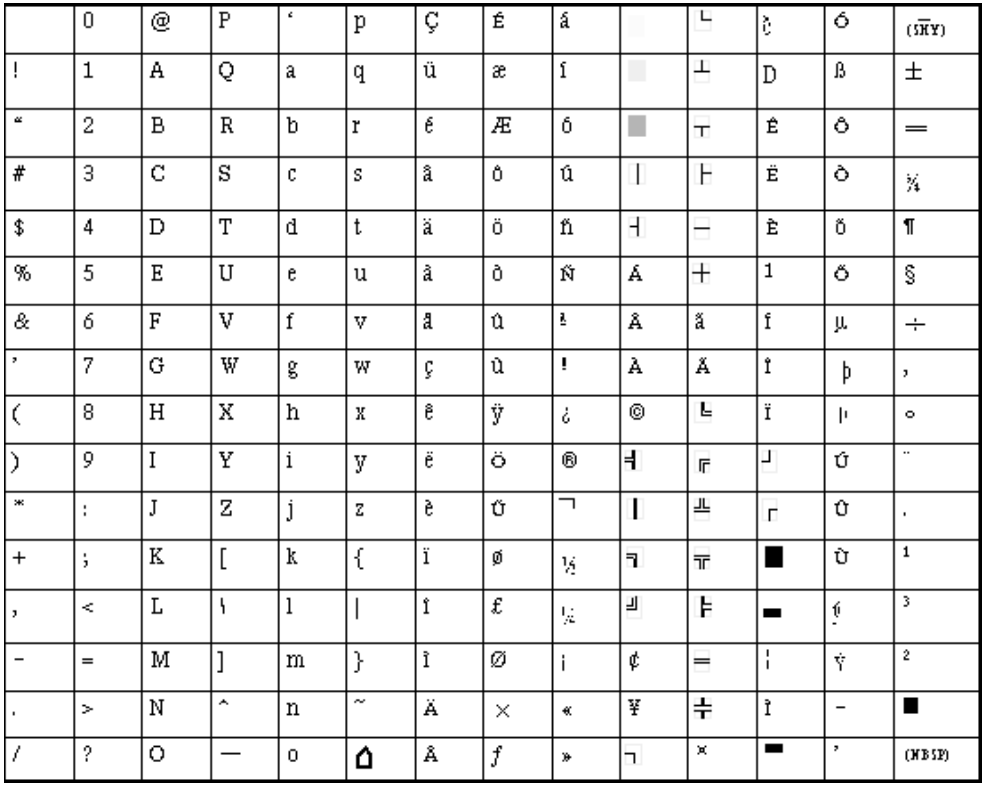

Figure 2–3: The Character Set Of the IBM850 Code Page

Figure 2–4 shows the character set of the ISO8859–1 code page

	0	@	P	,	p			(RSP)	°	À	Ð	â	ð
!	1	A	Q	a	q			¡	±	Á	Ñ	á	ñ
"	2	B	R	b	r			¢	²	Â	Ò	â	ò
#	3	C	S	c	s			£	³	Ã	Ó	ã	ó
$	4	D	T	d	t			¤	´	Ä	Ô	ä	ô
%	5	E	U	e	u			¥	µ	Å	Õ	å	õ
&	6	F	V	f	v			¦	¶	Æ	Ö	æ	ö
'	7	G	W	g	w			§	·	Ç	×	ç	÷
(	8	H	X	h	x			¨	¸	È	Ø	è	ø
)	9	I	Y	i	y			©	¹	É	Ù	é	ù
*	:	J	Z	j	z			ª	º	Ê	Ú	ê	ú
+	;	K	[	k	{			«	»	Ë	Û	ë	û
,	<	L	\	l	|			¬	¼	Ì	Ü	ì	ü
-	=	M	]	m	}			(SHY)	½	Í	Ý	í	ý
.	>	N	^	n	~			®	¾	Î	Þ	î	þ
/	?	O	_	o				¯	¿	Ï	ß	ï	ÿ

Figure 2–4: The Character Set Of the ISO8859–1 Code Page

2.2 Environments With Multiple Code Pages

In an application, not every component must have the same code page. Often components have different code pages. Sometimes every component has a different code page. Figure 2–5 shows a Russian application where each component has a different code page.

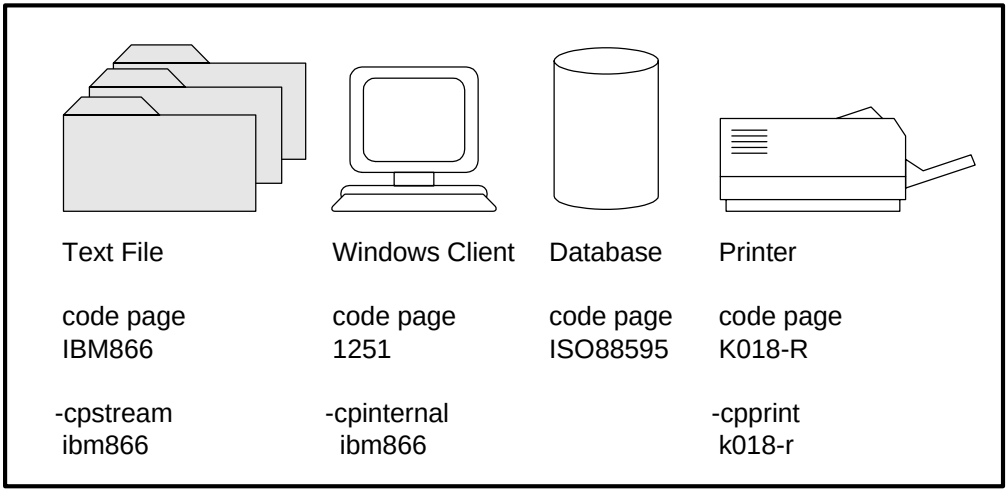

Figure 2–5: A Russian Application Where Each Component Has a Different Code Page

Figure 2–5 also shows startup parameters, each of which tells Progress the code page of the corresponding component. For more information on using startup parameters, see Chapter 10, "Deployment and Configuration."

2.3 Code-page Conversion

As data flows between application components that have different code pages, Progress sometimes converts the data from the code page of the source component to the code page of the destination component. This is illustrated in Figure 2–6.

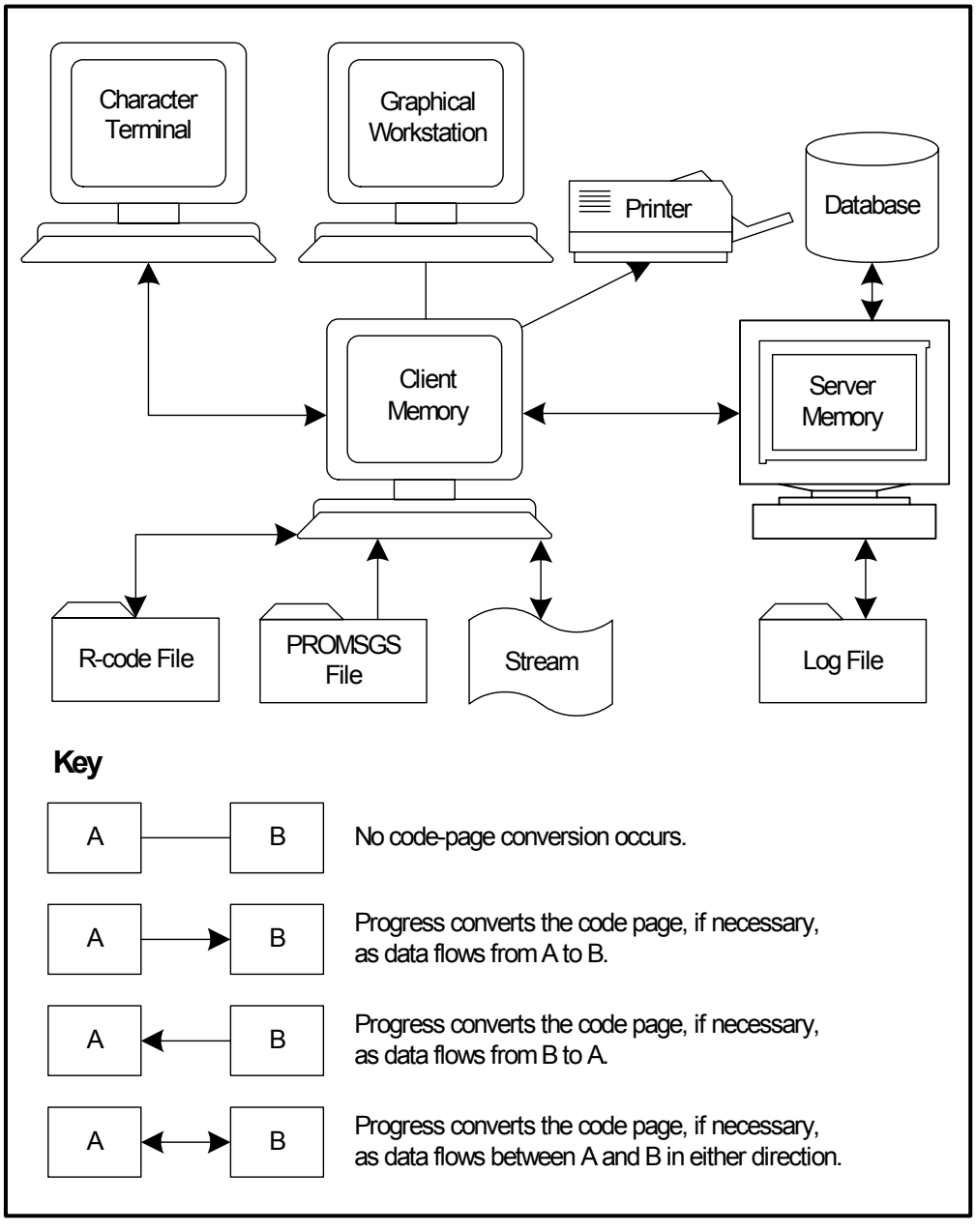

Figure 2–6: Code-page Conversion

For each line in Figure 2–6, Progress might:

- Convert the code page no matter which direction the data is flowing in
- Convert the code page if the data is flowing in one direction, but not the other
- Not convert the code page

To convert data from one code page to another, Progress uses a code-page conversion table. Progress supplies a collection of code-page conversion tables in the `DLC/prolang` directory. You can also create your own code-page conversion tables. For more information on code-page conversion tables, see Chapter 3, "Understanding Character Processing Tables." For more information on code-page conversion and databases, see Chapter 6, "Using Databases." For more information on code-page conversion and SQL-92, see Chapter 7, "Using SQL-92."

2.3.1 Streams and Code-page Conversion

Figure 2–6 shows a stream. Streams can include the following application components:

- Character terminals and emulators of character terminals
- Table dump (`.d`) and data definition (`.df`) files
- Files accessed by INPUT FROM or OUTPUT TO statements
- Operating system files that a Progress application reads or writes
- LISTING, XREF, and PREPROCESS files produced by the compiler

As Figure 2–6 shows, to decide whether or not to convert the code page of data flowing between two application components, Progress considers whether one of the application components is a stream. For more information on streams, see the *Progress Programming Handbook*.

Streams and the PROTERMCAP File

Certain streams—namely, character terminals, emulators of character terminals, and files accessed by INPUT FROM and OUTPUT TO statements—sometimes undergo a completely different kind of character conversion, one related to the PROTERMCAP file. This file, used by Progress applications that run on UNIX and that have a character interface, associates generic cursor-movement commands with the control codes required by a particular terminal.

Do not confuse data conversion related to code-page differences with data conversion related to the PROTERMCAP file. For more information on the PROTERMCAP file, see the *Progress Client Deployment Guide*.

2.3.2 Valid and Invalid Code-page Conversions

Sometimes converting from one code page to another garbles the data. This section states rules for determining if a particular code-page conversion garbles the data. In these rules, *source code page* means the code page you are converting from, *target code page* means the code page you are converting to, and a *valid* code-page conversion is one that does not garble the data. The rules are:

- A code-page conversion is valid if each character that appears in the source code page appears in the target code page.

  For example, it is valid to convert from the IBM850 code page (used for Latin-alphabet languages of western Europe and the Americas) to the ISO8859–1 code page (also used for Latin-alphabet languages of western Europe and the Americas) because each character that appears in IBM850 appears in ISO8859–1. Similarly, it is not valid to convert from the CP949 code page (used for Korean) to the KSC5601 code page (also used for Korean) because CP949 contains characters that KSC5601 does not.

- A code-page conversion is valid if every character that appears in the data appears in the target coded page.

  Sometimes, although one or more characters in the source code page do not appear in the target code page, none of these troublesome characters appears in the data. For example, suppose that you want to convert data from one code page to another, and that the source and target code pages are identical except that the source code page contains the Euro currency character while the target code page does not. If the Euro currency symbol does not appear in the data, this code-page conversion is valid.

  To check a text file for the presence of a particular character, use the PROUTIL utility with the CONVCHAR CHARSCAN qualifier. For more information on this technique, see Chapter 6, “Using Databases.”

2.3.3 Sockets and Code-page Conversion

If two applications using different code pages communicate using sockets (one of the external programming interfaces Progress provides), code-page conversion must be performed by the application. It is not performed automatically by the sockets layer. Specifically, it is not performed by socket’s WRITE() method or READ() method.

To perform the code-page conversion, use the 4GL CODEPAGE–CONVERT function either before invoking the 4GL PUT–STRING function or after invoking the 4GL GET–STRING function. For more information on using sockets in Progress applications, see the *Progress External Program Interfaces* manual.

2.4 The Undefined Code Page

A special code-page name, undefined, tells Progress not to do any conversions when reading or writing data to or from this code page. You can specify this name wherever you can specify a code-page name. While declaring data to be undefined is useful in certain situations, do not declare all of your data to be undefined, even if your environment is homogeneous. By identifying the code page of your data, you prepare Progress to handle correctly any future extensions to your environment.

For example, the code pages of the demo and sports databases are set to undefined so that you can use these databases with any character set. This is possible because they contain only ASCII characters, which are included in most other character sets. These databases also have their own character sorting for text within the database. If you elect to customize these databases to your code-page environment, convert the databases to your own code page and collation. For instructions on how to convert the databases to your own code pages, see Chapter 6, "Using Databases."

NOTE: If you set -cpinternal to undefined, all code-page conversions are disabled.

NOTE: If you set -cpinternal to undefined for a double-byte or UTF–8 client, multi-byte characters will not be treated properly. They will be treated as single-byte values, and data corruption is likely. The lead and trail bytes might convert as single-byte characters, the double-byte characters might be split, and phrases might be separated incorrectly.

2.5 Figuring Out the Code Page an Application Component Uses

To include a particular database, terminal, printer, or file in an application, you might have to figure out which code page the application component uses, which is not always intuitively obvious. This section describes some techniques for figuring this out.

2.5.1 7-bit Character Data

If you know for certain that some character data contains only 7-bit characters and you know the code page of your current environment, place commands that have the following syntax into `DLC/startup.pf`, the parameter file that every local Progress executable reads at startup:

SYNTAX

```
-cpinternal codepage-name
-cpstream  codepage-name
```

codepage-name

The code page of your current environment.

2.5.2 8-bit Character Data

To determine the code page of 8-bit character data, consider each place the data is stored or generated and determine the code page of each such place. The actual technique varies depending on the data location and is described separately by data location.

Some of the following techniques involve placing data into a text file. Use the `chkdotd.p` procedure (located in the `DLC/prolang` directory) to examine the contents of the file. Before you run this procedure, make sure that you start Progress with the -cpinternal startup parameter set to undefined. The `chkdotd.p` procedure examines all characters in the text file. For each character, `chkdotd.p` writes the character's numeric value and the number of times it appears. You then compare the output of `chkdotd.p` to printed charts of the various code pages used in your locale.

The standard approach is to take the numeric value of a character and see how many times it appears in the file. You then use the numeric value to index the character against a specific code page. If the corresponding character displayed in the code page is not a character that you would expect to see, or if the character occurs more often than it should, you can probably rule out that code page.

Techniques are described for the following data locations:

- Progress databases
- Character terminals
- Windows screen and keyboards
- Printers
- Table dump (`.d`) files and other external text files
- Progress libraries

2.5.3 Progress Databases

There are several techniques for determining the code page of a Version 8 or Version 9 database. Here is one technique:

1 ♦ Start the Procedure Editor.

2 ♦ Within the Procedure Editor, start the Data Dictionary utility, connect to the database, then exit the Data Dictionary utility.

3 ♦ From the Procedure Editor main menu, select the following:

Tools→ Data Administration→ Utilities→ Information

The Session Information window opens.

4 ♦ In the Session Information window, search for the Database Code Page label.

Immediately to its right, the name of the database's code page appears.

Another technique is to examine the metaschema field, _Db._Db-xl-name. To display this field, run the following code fragment:

```
FIND FIRST _Db.
DISPLAY _Db._Db-Xl-Name.
```

Yet another technique is to use the DBCODEPAGE function in a procedure, as shown in the following code fragment:

```
DEFINE VARIABLE i AS INTEGER.
REPEAT i=1 TO NUM-DBS:  /* For every connected database */
  DISPLAY DBCODEPAGE(i).
END.
```

For more information on the DBCODEPAGE function, see the *Progress Language Reference*.

2.5.4 Character Terminals

To determine the code page of a character terminal, use one of the following techniques:

- Read the terminal's documentation.
- Use the terminal's setup screens. Many terminals have setup screens that let you view or modify the code page.

Both of these techniques have one drawback: If your operating system contains device drivers that perform code-page conversions, they might convert the code page from that of the operating system to a new one. If this occurs, Progress does not see the code page of the operating system, but rather the new one. You must determine which code page Progress actually sees.

To determine which code page Progress actually sees, follow these steps:

1 ♦ Start Progress with -cpinternal set to undefined and set the TERM environment variable to disable PROTERMCAP IN/OUT mapping.

2 ♦ Execute the following code fragment:

```
REPEAT:
   READKEY.
   DISPLAY LASTKEY CHR(LASTKEY).
END.
```

3 ♦ Press several keys, including keys that correspond to the 8-bit extended characters. For each keystroke, Progress displays the 8-bit numeric value received for that key and echoes the character back to the terminal. If the character does not echo properly, your system is configured incorrectly. If the character does echo properly, the numeric value Progress displays is the 8-bit value for that character.

Once you have several such pairs of characters and numbers, compare them against likely code pages to determine the correct code page.

2.5.5 Windows Screen and Keyboard

The code page used for the Windows screen and keyboard is configurable by the user. To find the settings, look at the regional settings in the control panel window.

To test individual characters with the **ALT** key and the numeric keypad, follow these steps:

1 ♦ On Windows, start a simple application, such as Notepad, that lets you enter characters.

2 ♦ Press and hold **ALT**, then press 0 on the numeric keypad, followed by a three-digit decimal number on the numeric keypad, for a total of four digits.

3 ♦ Release the **ALT** key. The character that corresponds to the three-digit decimal number appears on the screen.

4 ♦ Repeat Steps 3 and 4 to get several pairs of characters and numbers.

5 ♦ Compare your character-number pairs against likely code pages to determine the correct code page.

Another way to get a list of character-number pairs to compare against a likely code page is to run Progress on Windows, set -cpinternal to undefined, and run the following code fragment:

```
REPEAT:
  READKEY.
  DISPLAY LASTKEY CHR(LASTKEY).
END.
```

2.5.6 Printers

To determine the code page a printer uses, check the documentation that comes with the printer and with the operating system.

You can also choose several characters from the code page you think the printer uses and determine the numeric values for those characters. Start Progress with -cpprint set to undefined, then run the following code fragment:

```
DEFINE VARIABLE i AS INTEGER.
OUTPUT TO PRINTER NO-MAP.
REPEAT i = 1 to 255:
    DISPLAY i CHR(i).
END.
OUTPUT CLOSE.
```

Examine the output and compare it with likely code pages, or try different settings of -cpprint until the output is correct.

2.5.7 Table Dump (.d) Files and Other External Text Files

If you know which 8-bit characters your data contains and can locate them in the table dump (.d) file, examine the numeric values of the data bytes in the file. You can do this with any hexadecimal dump program provided by your operating system or with a Progress program that has the following syntax:

SYNTAX

```
INPUT FROM filename NO-MAP.
REPEAT:
  READKEY.
  DISPLAY LASTKEY.
END.
```

filename

The name of your file.

NOTE: Start Progress with -cpinternal set to undefined.

2.5.8 Progress Libraries

To determine the code page a library is encoded in, use the PROLIB utility with the -list qualifier. The syntax is:

SYNTAX

```
prolib library-name -list
```

library-name

Specifies the name of a library. The library name must have a .pl extension.

For the complete syntax of the PROLIB utility, see the *Progress Client Deployment Guide*.

3

Understanding Character Processing Tables

Progress provides tables that let you tailor the character processing of applications to be deployed across multiple locales to the precise needs of each locale. These *character processing tables* include character attribute tables, case tables, collation tables, code-page conversion tables, and word-break tables.

These tables (except for word-break tables) are included in the `convmap.dat` file, which you compile to produce a `convmap.cp` file, which you provide access to by copying to a certain directory or by setting an environment variable. Progress supplies a default `convmap.dat` file and a corresponding default `convmap.cp` file.

Word-break tables, which the `convmap.dat` file does not include, reside in the `DLC/prolang/convmap` directory. You can create a word-break table from scratch or modify one that Progress supplies. In either case, you compile the word-break table, provide access to the compiled version of the word-break table by copying it to a certain directory or by setting an environment variable, associate the compiled version with a database, and rebuild the database's indexes.

This chapter have the following sections:

- The convmap.dat File and Its Tables
- Word-break Tables

NOTE: For a thorough explanation of code pages, see Chapter 2, "Understanding Code Pages."

3.1 The convmap.dat File and Its Tables

The convmap.dat file supplied by Progress, which resides in the DLC/prolang/convmap directory, does not contain character processing tables. Rather, it contains INCLUDE directives, each of which includes a file containing character processing tables. The included files also reside in the DLC/prolang/convmap directory.

NOTE: The convmap.dat file can contain INCLUDE directives and actual character processing tables (except for word-break tables) in any combination.

A typical included file is arabic.dat, which contains tables for locales that use the Arabic language. Table 3–1 lists the contents of arabic.dat.

Table 3–1: Contents Of arabic.dat *(1 of 2)*

Code Page	Character Processing Tables
1256	Character attribute table BASIC case table ARABIC9 collation table BASIC collation table
709	BASIC case table BASIC collation table Character attribute table Table for converting from code page 1256 to code page 709
708	BASIC case table BASIC collation table Character attribute table Table for converting from code page 1256 to code page 708
721	BASIC case table BASIC collation table Character attribute table Table for converting from code page 1256 to code page 721

Table 3–1: Contents Of arabic.dat *(2 of 2)*

Code Page	Character Processing Tables
711	BASIC case table BASIC collation table Character attribute table Table for converting from code page 1256 to code page 711
786	BASIC case table BASIC collation table Character attribute table Table for converting from code page 1256 to code page 786
714	BASIC case table BASIC collation table Character attribute table Table for converting from code page 1256 to code page 714
710	BASIC case table BASIC collation table Character attribute table Table for converting from code page 1256 to code page 710
720	BASIC case table BASIC collation table Character attribute table Table for converting from code page 1256 to code page 720

Table 3–1 shows several characteristics of the `arabic.dat` file in particular and of character processing tables in general:

- The `arabic.dat` file contains tables for a variety of code pages.
- These tables consist of character attribute tables, case tables, collation tables, and code-page conversion tables.
- Each table applies to a particular code page.
- Case tables and collation tables have names, while character attribute tables and code-page conversion tables do not.
- A code page can have only one character attribute table, only one code-page conversion table for a given code-page conversion, only one case table with a particular name, and only one collation table with a particular name.
- Multiple code pages might have collation tables or case tables with the same name. For example, code page 710 and code page 720 each have a case table named BASIC, as Table 3–1 shows.
- Every code page does not have every kind of character processing table.

3.1.1 Character Attribute Tables

A character attribute table tells Progress whether an element of a code page represents a character or not. Non-characters include numerals, punctuation, spaces, and carriage returns.

Figure 3–1 shows code page 1256's character attribute table, the first table in the `arabic.dat` file.

```
# This table contains the attributes for code page 1256
CODEPAGE
CODEPAGE-NAME "1256"
TYPE "1"
ISALPHA
  /*000-015*/  000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
  /*016-031*/  000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
  /*032-047*/  000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
  /*048-063*/  000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
  /*064-079*/  000 001 001 001 001 001 001 001 001 001 001 001 001 001 001 001
  /*080-095*/  001 001 001 001 001 001 001 001 001 001 001 000 000 000 000 000
  /*096-111*/  000 001 001 001 001 001 001 001 001 001 001 001 001 001 001 001
  /*112-127*/  001 001 001 001 001 001 001 001 001 001 001 000 000 000 000 000
  /*128-143*/  000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
  /*144-159*/  000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
  /*160-175*/  000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
  /*176-191*/  000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
  /*192-207*/  000 001 001 001 001 001 001 001 001 001 001 001 001 001 001 001
  /*208-223*/  001 001 001 001 001 001 001 000 001 001 001 001 000 001 001 001
  /*224-239*/  000 001 000 001 001 001 001 000 000 000 000 000 001 001 000 000
  /*240-255*/  001 001 001 001 000 001 001 000 001 000 001 000 000 000 000 000
ENDTABLE
ENDCODEPAGE
```

Figure 3–1: Code Page 1256's Character Attribute Table

A character processing table contains a value for each element in the code page. The values are arranged in rows of sixteen. The first value in the first row corresponds to the first element (element 0), the second value in the first row corresponds to the second element (element 1), the first value in the second row corresponds to the seventeenth element (element 16), and the last value in the last row corresponds to the last element (element 255).

The value 1 means the corresponding element is alphabetic, while the value 0 means the corresponding element is not alphabetic. Figure 3–1 shows that elements 67, 210, and 250 are alphabetic, while elements 48, 63, and 238 are non-alphabetic.

3.1.2 Case Tables

A case table tells Progress how to convert a character in the code page from uppercase to lowercase or from lowercase to uppercase. Progress uses a case table when it encounters code such as:

- The Progress 4GL CAPS and LC functions
- The Progress SQL-89 UPPER and LOWER functions

Figure 3–2 shows code page ISO8859–15's BASIC case table, which resides in the 8859-15.dat file.

```
# Case tables for code page ISO8859-15 and case table basic
CASE
CODEPAGE-NAME ISO8859-15
CASETABLE-NAME BASIC
TYPE 1
UPPERCASE-MAP
  /*000-015*/  000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015
  /*016-031*/  016 017 018 019 020 021 022 023 024 025 026 027 028 029 030 031
  /*032-047*/  032 033 034 035 036 037 038 039 040 041 042 043 044 045 046 047
  /*048-063*/  048 049 050 051 052 053 054 055 056 057 058 059 060 061 062 063
  /*064-079*/  064 065 066 067 068 069 070 071 072 073 074 075 076 077 078 079
  /*080-095*/  080 081 082 083 084 085 086 087 088 089 090 091 092 093 094 095
  /*096-111*/  096 065 066 067 068 069 070 071 072 073 074 075 076 077 078 079
  /*112-127*/  080 081 082 083 084 085 086 087 088 089 090 123 124 125 126 127
  /*128-143*/  128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
  /*144-159*/  144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159
  /*160-175*/  160 161 162 163 164 165 166 167 166 169 170 171 172 173 174 175
  /*176-191*/  176 177 178 179 180 181 182 183 184 185 186 187 188 188 190 191
  /*192-207*/  192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207
  /*208-223*/  208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223
  /*224-239*/  192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207
  /*240-255*/  208 209 210 211 212 213 214 247 216 217 218 219 220 221 222 190
ENDTABLE
LOWERCASE-MAP
  /*000-015*/  000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015
  /*016-031*/  016 017 018 019 020 021 022 023 024 025 026 027 028 029 030 031
  /*032-047*/  032 033 034 035 036 037 038 039 040 041 042 043 044 045 046 047
  /*048-063*/  048 049 050 051 052 053 054 055 056 057 058 059 060 061 062 063
  /*064-079*/  064 097 098 099 100 101 102 103 104 105 106 107 108 109 110 111
  /*080-095*/  112 113 114 115 116 117 118 119 120 121 122 091 092 093 094 095
  /*096-111*/  096 097 098 099 100 101 102 103 104 105 106 107 108 109 110 111
  /*112-127*/  112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127
  /*128-143*/  128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
  /*144-159*/  144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159
  /*160-175*/  160 161 162 163 164 165 168 167 168 169 170 171 172 173 174 175
  /*176-191*/  176 177 178 179 184 181 182 183 184 185 186 187 189 189 255 191
  /*192-207*/  224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239
  /*208-223*/  240 241 242 243 244 245 246 215 248 249 250 251 252 253 254 223
  /*224-239*/  224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239
  /*240-255*/  240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255
ENDTABLE
ENDCASE
```

Figure 3–2: Code Page ISO8859–15's BASIC Case Table

Case tables have two sections, one for converting a character to uppercase and the other for converting a character to lowercase. Each section contains a value for each element in the code page. The values are arranged in rows of sixteen. In each section, the first value in the first row corresponds to the first element (element 0) of the code page, the second value in the first row corresponds to the second element (element 1), the first value in the second row corresponds to the seventeenth element (element 16), and the last value in the last row corresponds to the last element (element 255).

Within a section, each value is the number of the element with the opposite case. Figure 3–2 shows, for example, that the uppercase equivalent of element 97 is element 65 and that the lowercase equivalent of element 65 is element 97. In code page ISO8859–15, element 97 represents the character “a” and element 65 represents the character “A.” In other words, this case table tells us that the uppercase equivalent of “a” is “A” and that the lowercase equivalent of “A” is “a,” which agrees with what we know about the characters “A” and “a” in ISO8859–15.

NOTE: Languages that do not distinguish between uppercase and lowercase, such as Arabic and Hebrew, still have case tables. These case tables map each code page element to itself. This means that if an application tries to change the case of, say, an Arabic character string, the result is the same character string.

3.1.3 Collation Tables

A collation table tells Progress how to sort or compare characters. Progress uses a collation table when it:

- Compares CHARACTER strings using a 4GL relational operator or the 4GL COMPARE function
- Computes the collation value of a CHARACTER data item using the 4GL COLLATE option of the FOR statement, the OPEN QUERY statement, and the PRESELECT phrase
- Sorts the results of a 4GL query that uses the FOR statement's BY or EACH option
- Sorts the results of an SQL-89 query that uses the SELECT statement's ORDER BY option
- Builds or rebuilds databases indexes

Figure 3–3 shows code page 1253's GREEK collation table, which resides in the `greek.dat` file.

```
COLLATION
CODEPAGE-NAME 1253
COLLATION-NAME GREEK
COLLATION-TRANSLATION-VERSION 1.0-16
CASE-INSENSITIVE-SORT
  /*000-015*/  000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015
  /*016-031*/  016 017 018 019 020 021 022 023 024 025 026 027 028 029 030 031
  /*032-047*/  032 033 034 035 036 037 038 039 040 041 042 043 044 045 046 047
  /*048-063*/  048 049 050 051 052 053 054 055 056 057 058 059 060 061 062 063
  /*064-079*/  064 065 069 073 077 081 085 087 091 097 101 103 107 111 115 121
  /*080-095*/  125 127 131 135 139 143 145 147 153 157 161 165 166 167 168 169
  /*096-111*/  170 065 069 073 077 081 085 087 091 097 101 103 107 111 115 121
  /*112-127*/  125 127 131 135 139 143 145 147 153 157 161 171 172 173 174 175
  /*128-143*/  176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191
  /*144-159*/  192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207
  /*160-175*/  208 209 067 210 211 212 213 214 215 216 217 218 219 220 221 222
  /*176-191*/  223 224 225 226 227 228 229 230 083 093 099 231 123 232 149 163
  /*192-207*/  099 067 071 075 079 083 089 093 095 099 105 109 113 117 119 123
  /*208-223*/  129 133 233 137 141 149 151 155 159 163 099 149 067 083 093 099
  /*224-239*/  149 067 071 075 079 083 089 093 095 099 105 109 113 117 119 123
  /*240-255*/  129 133 137 137 141 149 151 155 159 163 099 149 123 149 163 234
ENDTABLE
CASE-SENSITIVE-SORT
  /*000-015*/  000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015
  /*016-031*/  016 017 018 019 020 021 022 023 024 025 026 027 028 029 030 031
  /*032-047*/  032 033 034 035 036 037 038 039 040 041 042 043 044 045 046 047
  /*048-063*/  048 049 050 051 052 053 054 055 056 057 058 059 060 061 062 063
  /*064-079*/  064 065 069 073 077 081 085 087 091 097 101 103 107 111 115 121
  /*080-095*/  125 127 131 135 139 143 145 147 153 157 161 165 166 167 168 169
  /*096-111*/  170 066 070 074 078 082 086 088 092 098 102 104 108 112 116 122
  /*112-127*/  126 128 132 136 140 144 146 148 154 158 162 171 172 173 174 175
  /*128-143*/  176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191
  /*144-159*/  192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207
  /*160-175*/  208 209 067 210 211 212 213 214 215 216 217 218 219 220 221 222
  /*176-191*/  223 224 225 226 227 228 229 230 083 093 099 231 123 232 149 163
  /*192-207*/  100 067 071 075 079 083 089 093 095 099 105 109 113 117 119 123
  /*208-223*/  129 133 233 137 141 149 151 155 159 163 099 149 068 084 094 100
  /*224-239*/  150 068 072 076 080 084 090 094 096 100 106 110 114 118 120 124
  /*240-255*/  130 134 138 138 142 150 152 156 160 164 100 150 124 150 164 234
ENDTABLE
ENDCOLLATION
```

Figure 3–3: Code Page 1253's GREEK Collation Table

Figure 3–3 shows that the collation table has two sections, one for case-insensitive sorts and one for case-sensitive sorts. Each section has a value for each element in the code page. The value of an element represents the sort order of that element. For example, in a case-insensitive sort that uses this collation table, element 1 sorts first and element 97 sorts sixty-sixth.

3.1.4 Code-page Conversion Tables

A code-page conversion table tells Progress how to convert a character on one code page to the equivalent character on another code page. Progress uses a code-page conversion table when:

- You convert a database and its data to a different code page. For more information on converting a database to a different code page, see Chapter 6, "Using Databases."
- Progress automatically converts data from one code page to another during execution. For more information on automatic code-page conversion, see Chapter 2, "Understanding Code Pages."

Figure 3–4 shows a code-page conversion table for converting from code page 1256 to code page 709. Code pages 1256 and 709 are used in locales that use the Arabic language. This table resides in the file `arabic.dat`.

```
# This contains the data needed to convert from
# codepage 1256 to ASMO-449+ codepage 709
CONVERT
SOURCE-NAME "1256"
TARGET-NAME "709"
TYPE "1"
  /*000-015*/  000 001 249 003 004 005 006 007 008 009 010 011 012 013 014 015
  /*016-031*/  016 017 018 019 022 023 024 025 026 027 028 029 030 031 254 255
  /*032-047*/  032 033 034 035 036 037 038 039 040 041 042 043 044 045 046 047
  /*048-063*/  048 049 050 051 052 053 054 055 056 057 058 059 060 061 062 063
  /*064-079*/  064 065 066 067 068 069 070 071 072 073 074 075 076 077 078 079
  /*080-095*/  080 081 082 083 084 085 086 087 088 089 090 091 092 093 094 095
  /*096-111*/  096 097 098 099 100 101 102 103 104 105 106 107 108 109 110 111
  /*112-127*/  112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127
  /*128-143*/  128 132 174 159 134 141 142 143 144 145 146 190 148 149 155 156
  /*144-159*/  157 158 160 161 162 002 163 224 164 165 166 188 167 252 168 169
  /*160-175*/  171 172 154 176 177 178 179 021 180 181 182 183 184 173 185 186
  /*176-191*/  189 192 220 221 222 223 020 250 153 243 187 244 245 246 247 191
  /*192-207*/  248 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207
  /*208-223*/  208 209 210 211 212 213 214 170 215 216 217 218 219 225 226 227
  /*224-239*/  133 228 131 229 230 231 232 135 138 130 136 137 233 234 140 139
  /*240-255*/  235 236 237 238 147 239 240 175 241 151 242 150 129 251 152 253
ENDTABLE
ENDCONVERT
```

Figure 3–4: Table For Converting From Code Page 1254 To Code Page 709

A code-page conversion table contains a value for every element in the code page. Values appear in rows of sixteen. Each value represents the corresponding element in the target code page. For example, as Figure 3–4 shows, element 240 in code page 1256 corresponds to element 235 in code page 709.

Converting To and From UTF–8

If you add a new code-page conversion table that converts to UTF–8 (an encoding of Unicode), you must set the correct value for TYPE. As shown in Figure 3–4, this number appears at the top of the code-page conversion table, to the right of the literal TYPE, in the double quotes.

The value you supply for TYPE depends on whether the non-UTF-8 code page is single byte or double byte, as Table 3–2 shows.

Table 3–2: Converting to UTF–8

Conversion	Value for TYPE
From a single-byte code page to UTF–8	19
From a double-byte code page to UTF–8	17

For example, to convert from ISO8859–1, a single-byte code page, to UTF–8, set TYPE to 19.

When you compile code-page conversion tables for converting to UTF–8, Progress automatically computes the inverse conversions and assigns the correct value for TYPE, as shown in Table 3–3.

Table 3–3: Converting From UTF–8

Conversion	Value for TYPE
From UTF–8 to a single-byte code page	20
From UTF–8 to a double-byte code page	18

For more information on multi-byte code pages, see Chapter 8, “Using Multi-byte Code Pages,” and Chapter 9, “Using Unicode.”

3.1.5 Modifying convmap.dat Or a File It Includes

You can modify the `convmap.dat` file to add or delete INCLUDE directives. You might delete INCLUDE directives for files that do not apply to your target locales, or add INCLUDE directives for additional files you create for your target locales.

The syntax of the INCLUDE directive is:

SYNTAX

```
INCLUDE
INCLUDE-FILE include-file-name.dat
```

include-file-name.dat

The name of the file being included.

In addition to modifying the `convmap.dat` file, you might add new tables to an existing file. To do so, copy one of the existing tables to use as a template, then modify the template as required. Be sure to change the name of the table as required.

For more information on the format of files with the `.dat` extension, see Appendix B, "Character Processing Table Formats."

NOTE: Modifying collation tables involves additional steps. For more information on modifying collation tables, see the "Modifying Collation Tables" section.

3.1.6 Compiling the convmap.dat File

You must recompile the `convmap.dat` file if you modify it or a file it includes.

To compile the `convmap.dat` file, run the PROUTIL utility with the CODEPAGE-COMPILER option. The syntax is:

SYNTAX

```
proutil -C CODEPAGE-COMPILER sourcefile.dat outputfile.cp
```

sourcefile.dat

The `convmap.dat` file (or file with an equivalent format).

outputfile.cp

The `convmap.cp` file (or file with an equivalent format).

For the complete syntax of the PROUTIL utility, see the *Progress Database Administration Guide and Reference*.

3.1.7 Providing Access To the convmap.cp File

To tell Progress where to find the convmap.cp file, use one of the following techniques:

- Copy the `convmap.cp` file to the Progress installation directory.
- Set the PROCONV environment variable to point to the `convmap.cp` file.
- Use the -convmap startup parameter. For example:

3.1.8 Modifying Collation Tables

Modifying collation tables is somewhat different from modifying the other non-word-break character-processing tables. You must first dump the collation table, edit it, load the modified table, and rebuild the indexes.

Preliminary Considerations

Before you modify the database's collation tables, check the `DLC/prolang` directory for region-specific data definition (`.df`) files. These are database collation files that can be loaded into an empty database. Progress provides a collection of data definition files, one of which might suit your collation needs. For example, the `DLC/prolang/ger` directory contains the German-specific data definition files `ger850.df` and `ger8859.df`.

To modify how Progress performs 4GL comparisons, start Progress against the database that contains collation tables you want, or use the -cpcoll startup parameter to point to the tables you want. However, -cpcoll can point only to tables that reside in the `convmap.cp` file. If the collation/code-page pair you want does not reside in `convmap.cp`, you can build your own collation table, as described in the "Editing the Collation Table" section of this chapter.

Dumping a Collation Table

To dump a collation table, follow these steps:

1 ♦ In the Data Administration tool, choose Admin→ Dump Data and Definitions→ Collation. The Dump Collation Tables dialog box appears.

2 ♦ Type a dump name in the Output File field (`_tran.df` is the default), then choose OK.

3 ♦ Enter the name of the code page the file will be written out with.

The `_tran.df` file contains the collation table for your database. Figure 3–5 shows a sample `_tran.df` file.

```
UPDATE DATABASE "?"
  COLLATION-TRANSLATION-VERSION 1.0-16
  COLLATION-NAME "basic"
  INTERNAL-EXTERNAL-TRAN-TABLE
  ?
  EXTERNAL-INTERNAL-TRAN-TABLE
  ?
  CASE-INSENSITIVE-SORT
  /*000-015*/  000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015
  /*016-031*/  016 017 018 019 020 021 022 023 024 025 026 027 028 029 030 031
  /*032-047*/  032 033 034 035 036 037 038 039 040 041 042 043 044 045 046 047
  /*048-063*/  048 049 050 051 052 053 054 055 056 057 058 059 060 061 062 063
  /*064-079*/  064 065 066 067 068 069 070 071 072 073 074 075 076 077 078 079
  /*080-095*/  080 081 082 083 084 085 086 087 088 089 090 091 092 093 094 095
  /*096-111*/  096 065 066 067 068 069 070 071 072 073 074 075 076 077 078 079
  /*112-127*/  080 081 082 083 084 085 086 087 088 089 090 123 124 125 126 127
  /*128-143*/  176 177 178 179 180 185 186 187 188 191 192 193 194 195 196 197
  /*144-159*/  200 201 202 203 204 205 206 213 217 218 219 220 223 242 000 158
  /*160-175*/  255 173 189 156 207 190 221 245 249 184 229 174 199 240 199 238
  /*176-191*/  248 241 253 252 239 230 244 250 247 251 229 175 172 171 243 168
  /*192-207*/  065 065 065 065 065 143 146 067 069 069 069 069 073 073 073 073
  /*208-223*/  068 078 079 079 079 079 079 158 157 085 085 085 085 089 222 000
  /*224-239*/  065 065 065 065 065 143 146 067 069 069 069 069 073 073 073 073
  /*240-255*/  068 078 079 079 079 079 079 246 157 085 085 085 085 089 222 089
  CASE-SENSITIVE-SORT
  /*000-015*/  000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015
                                          .
                                          .
                                          .

  /*240-255*/  208 110 111 111 111 228 111 246 155 117 117 117 117 121 231 121
.
PSC
cpstream=ibm850
.
0000005361
```

Figure 3–5: Sample _tran.df File

NOTE: The IBM850 code page that appears in the trailer of the sample `_tran.df` file in Figure 3–5 is not the code page name of the database. It is the code page that was used to write out the data definition (`.df`) file.

Editing the Collation Table

As you read this section, you can refer to Figure 3–3, which shows a Greek collation table.

To modify the CASE-INSENSITIVE-SORT and CASE-SENSITIVE-SORT tables, use your favorite text editor.

A collation table for a single-byte code page must provide 256 values in 16 rows of 16 cells. The number you provide for each cell is a three-digit decimal. A collation name cannot exceed 19 characters and can include the characters A–Z, a–z, 0–9, and the dash (-).

The keyword COLLATION tells Progress that the following table entry is for 4GL comparisons. The keyword CODEPAGE-NAME specifies the name of the code page that the collations are for. The keyword COLLATION-NAME specifies a name for the collation.

At run time, Progress searches the `convmap.cp` file (or an equivalent file) to locate the correct collation tables. As a key to these tables, Progress uses the collation name (specified by -cpcoll) and a code page name (specified by -cpinternal). The names specified for these parameters must match the names in the `convmap.cp` file. These names are case insensitive.

The keyword COLLATION-TRANSLATION-VERSION specifies a value that Progress uses internally. Progress Software Corporation recommends that you specify the value 1.1-16 for single-byte collations.

The CASE-INSENSITIVE-SORT and CASE-SENSITIVE-SORT tables are identical to those used for a database and operate the same way. You modify them in the same way. For information on how to modify these tables, see the "Modifying Collation Tables" section.

The ENDTABLE keyword tells Progress that the table is finished. The ENDCOLLATION keyword tells Progress that the collation entry is complete.

Loading the Modified Collation Table

To load the modified tables back into the database, follow these steps:

1 ♦ In the Data Administration tool, choose Admin→ Load Data and Definitions→ Data Definitions (`.df` file). The Load Data Definitions dialog box appears.

2 ♦ Type the name of the data definition file that you modified (the default name is `_tran.df`) in the Input File field, then choose OK.

Rebuilding the Indexes

The Data Administration tool loads the schema information into the database. Before the database can use these tables, you must run the rebuild the database's indexes. To do so, run the PROUTIL utility with the IDXBUILD modifier.

NOTE: The database cannot be in use when you do this.

The syntax of the PROUTIL utility with the IDXBUILD qualifier is:

SYNTAX

```
proutil db-name -C idxbuild all
```

For the complete syntax of the PROUTIL utility, see the *Progress Database Administration Guide and Reference*.

Modifying, Compiling, and Providing Access To the CONVMAP File

You must still:

- Modify the `convmap.dat` file. For more information, see the "Modifying convmap.dat Or a File It Includes" section.
- Compile the `convmap.dat` file. For more information, see the "Compiling the convmap.dat File" section.
- Provide access to the resulting `convmap.cp` file. For more information, see the "Providing Access To the convmap.cp File" section.

3.1.9 Finding Additional Information On Character Processing Tables Other Than Word-break Tables

For more information on character attribute tables, case tables, collation tables, and code-page conversion tables, check these cross-references:

- See Chapter 10, "Deployment and Configuration," for information on setting startup parameters to tell Progress which code page, collation table, or case table to use.
- Run the procedure `list-cp.p` in the `DLC/prolang` directory to see a list of the collations Progress supports.
- Run the procedure `listconv.p` in the `DLC/prolang` directory to see a list of the code-page conversions Progress supports
- See the `readme` file in the `DLC/prolang` directory for information on:
  - The locale-specific subdirectories of the `DLC/prolang` directory
  - The case tables Progress provides
  - The collation tables Progress provides
  - The code-page conversion tables Progress provides

3.2 Word-break Tables

Progress uses word-break tables to process 4GL and SQL-89 queries that use the CONTAINS operator of the WHERE option of the record phrase.

The following is an example of a 4GL query that uses CONTAINS:

```
FOR EACH customer WHERE comments CONTAINS "credit hold":
  DISPLAY name cust-num comments.
END.
```

The following is an example of an SQL-89 query that uses CONTAINS:

```
SELECT name, cust-num, comments
  FROM customer
  WHERE comments CONTAINS "credit hold".
```

This chapter focuses on the use of word-break tables in international applications. For a general discussion of word-break tables, see the *Progress Programming Handbook*.

3.2.1 Why Progress Uses Word-break Tables

Progress uses word-break tables for the following reasons:

- Word-break tables are required by word indexing, which accelerates the processing of queries that use the CONTAINS operator.

  Consider the preceding examples. To process such queries quickly, Progress must quickly find the database records that contain the words of the target string (for example, “credit” and “hold”). To accelerate its searching, Progress uses word indexes. But before Progress can use word indexes, Progress must build them. To build them, Progress breaks each database field to be word indexed into separate words. To do this correctly, Progress needs to know which characters in the field act as word delimiters. This is the information word-break tables provide.

- Different locales can have different word-break conventions. For example, the character “!” might be a word-delimiter in one language and part of a word in another language.

- Different applications can have different word-break conventions. For example, an inventory application that uses part numbers might use the characters “@” and “.” as word delimiters that separate the different components of the part number (factory, floor, bin, assembly, subassembly, etc.), while an email application might consider these characters to be parts of words—in this case, email addresses.

3.2.2 Creating and Modifying Word-break Tables

Progress provides a collection of word-break tables in the DLC/prolang/convmap directory. Figure 3–6 shows one of them, big5-bas.wbt. Its name reflects the code page BIG-5, a code page used for Traditional Chinese.

```
/*
 *
 * NAME: big5-bas.wbt
 * Progress Word Break Source File for codepage big-5
 *
 */

version = 9
codepage = big-5
wordrules-name = basic
type = 3

/* Special word break rules table */
word_attr =
{
     '.',  BEFORE_DIGIT,   /* part of a word only if followed by a digit */
     ',',  BEFORE_DIGIT,
     '-',  BEFORE_DIGIT,
     ''',  IGNORE,         /* completely ignore it */
     '$',  USE_IT,         /* always part of a word */
     '%',  USE_IT,
     '#',  USE_IT,
     '@',  USE_IT,
     '_',  USE_IT
};
```

Figure 3–6: The big5-bas.wbt Word-break Table

Understanding Word-delimiter Attributes

The keywords BEFORE_DIGIT, IGNORE, and USE_IT, which appear in Figure 3–6, are word-delimiter attributes. Each word-delimiter attribute describes a word-break role played by a code page element. The complete set of word-delimiter attributes appears in Table 3–4.

Table 3–4: Word-delimiter Attributes *(1 of 2)*

Word Delimiter Attribute	Description	Default
LETTER	Always part of a word.	Assigned to all characters that the current attribute table defines as alphabetic. In English, these are the uppercase characters A–Z and the lowercase characters a–z.
DIGIT	Always part of a word.	Assigned to the characters 0–9.
USE_IT	Always part of a word.	Assigned to the following characters: • Dollar sign ($) • Percent sign (%) • Number sign (#) • At symbol (@) • Underline (_)
BEFORE_LETTER	Part of a word only if followed by a character with the LETTER attribute. Else, treated as a word delimiter.	–

Table 3–4: Word-delimiter Attributes *(2 of 2)*

Word Delimiter Attribute	Description	Default
BEFORE_DIGIT	Treated as part of a word only if followed by a character with the DIGIT attribute.	Assigned to the following characters: • Period (.) • Comma (,) • Hyphen (-) For example, "12.34" is one word, but "ab.cd" is two words.
BEFORE_LET_DIG	Treated as part of a word only if followed by a character with the LETTER or DIGIT attribute.	–
IGNORE	Ignored.	Assigned to the apostrophe (’). For example, "John’s" is equivalent to "Johns."
TERMINATOR	Word delimiter.	Assigned to all other characters.

Word-break Table Syntax

Word-break behavior varies widely between and even within locales. If CONTAINS queries do not work as expected in a particular locale, you can copy a word-break table that Progress provides and modify it as necessary. You can also create a word-break table from scratch. The syntax is as follows:

SYNTAX

```
[ #define symbolic-name symbol-value ] . . .

[ Version = 9
   Codepage = codepage-name
   wordrules-name = wordrules-name
   type = table-type
]

word_attr =
{
  { char-literal | hex-value | decimal-value } , word-delimiter-attribute
      [ , { char-literal | hex-value | decimal-value }
          , word-delimiter-attribute ] . . .
};
```

symbolic-name

The name of a symbol.

For example: DOLLAR-SIGN

symbol-value

The value of the symbol.

For example: ’$’

NOTE: Although some versions of Progress let you compile word-break tables that omit all items within the second pair of square brackets, Progress Software Corporation (PSC) recommends that you always include these items. If the source-code version of a compiled word-break table lacks these items, and the associated database is not so large as to make this impractical, PSC recommends that you add these items to the table, recompile the table, reassociate the table with the database, and rebuild the indexes.

codepage-name

The name, not surrounded by quotes, of the code page the word-break table is associated with. The maximum length is 20 characters.

For example: UTF–8

wordrules-name

The name, not surrounded by quotes, of the compiled word-break table. The maximum length is 20 characters.

For example: utf8sample

table-type

The number 3.

NOTE: Although Progress supports existing word-break tables of type 1 and type 2, Progress Software Corporation recommends that, if feasible, you change their table type to 3. If you do, you must also recompile the word-break table, reassociate it with the database, and rebuild the indexes.

char-literal

A character within single quotes or a `symbolic-name`, which represents a character in the code page.

For example: '#'

hex-literal

A hexadecimal value or a `symbolic-name`, which represents a character in the code page.

For example:0xAC

decimal-literal

A decimal value or a `symbolic-name`, which represents a character in the code page.

For example: 39

word-delimiter-attribute

In what context the character is a word delimiter. Use one of the word delimiter attributes in Table 3–4.

3.2.3 Compiling Word-break Tables

After you create or modify a word-break table, you must compile it using the PROUTIL utility with the WBREAK-COMPILER qualifier. The syntax is:

SYNTAX

```
proutil -C wbreak-compiler src-file rule-num
```

src-file

The name of the word-break table file to be compiled.

rule-num

A number between 1 and 255 inclusive that identifies this word-break table within your Progress installation.

The PROUTIL utility names the compiled version of the word-break table `proword.`*`rule-num`*. For example, if *`rule-num`* is 34, PROUTIL names the compiled version `proword.34`

For the complete syntax of the PROUTIL utility, see the *Progress Database Administration Guide and Reference*.

3.2.4 Providing Access To Word-break Tables

To allow database servers and shared-memory clients to access the compiled version of a word-break table, use one of the following techniques:

- Copy the compiled word-break table to the `DLC` directory.
- Set the `PROWD`*`rule-num`* environment variable to point to the compiled word-break table. For example, for a compiled word-break table that has the name `proword.34` and that resides in the `DLC/mydir/mysubdir` directory, set the environment variable `PROWD34` to `DLC/mydir/mysubdir/proword.34`.

  NOTE: Although the name of the compiled version of the word-break table has a dot, the name of the corresponding environment variable does not.

3.2.5 Associating Word-break Tables With Databases

After you compile a word-break table and provide access to the compiled version, you must associate the compiled version with a database using the PROUTIL utility with the WORD-RULES qualifier. The syntax is:

SYNTAX

```
proutil database -C word-rules rule-num
```

database

The name of the database.

rule-num

The value of *rule-num* you specified when you compiled the word-break table.

To associate the database with the default word-break rules, set *rule-num* to zero.

NOTE: Setting *rule-num* to zero associates the database with the default word-break rules for the current code page.

For the complete syntax of the PROUTIL utility, see the *Progress Database Administration Guide and Reference*.

3.2.6 Rebuilding the Indexes

For a database's word indexes to reflect changes in the word-break table, you must rebuild a database's indexes. To do so, use the PROUTIL utility with the IDXBUILD or IDXFIX qualifier.

NOTE: You can use IDXFIX when the database is online.

The syntax of PROUTIL with the IDXBUILD qualifier is:

SYNTAX

```
proutil db-name -C idxbuild [ all ]
     [ -T dir-name ] [ -TB blocksize ]
     [ -TM n] [ -B n]
```

The syntax of PROUTIL with the IDXFIX qualifier is:

SYNTAX

```
proutil db-name -C idxfix
```

For the complete syntax of the PROUTIL utility, see the *Progress Database Administration Guide and Reference*.

4

Preparing the Code

Writing applications for deployment in more than one locale poses challenges in several areas. This chapter addresses the general and specific issues involved in writing the applications.

This chapter contains the following sections:

- Guidelines and Methodology
- Input and Output
- Data-processing Issues
- Sorting Data
- Compiling Translated Applications

4.1 Guidelines and Methodology

An application that can potentially run worldwide follows certain application design and programming conventions that keep code as flexible as possible to accommodate the cultural, linguistic, technical, and legal differences between locales.

When designing your application and planning for its maintainability, you have to balance the economy of a single code source against the international customer requirements for localized applications with regional code sources.

The programming conventions affect how you write Progress code. These are the general guidelines to follow when programming an international application:

- Process character data as characters—not bytes.
- Use variables instead of hard coded values that might vary internationally. Make provisions for procedure substitution for instances when an algorithm changes for a particular locale (for example, tax laws).
- Code with translation in mind, prepare for string expansion, and identify string constants that should not be translated (for example, strings in queries and keywords).

The following sections discuss the general issue of how to structure your code and the specific programming conventions.

4.1.1 Structuring Source Code

By the time you have localized an application to address the needs of users in a variety of locales, you might find yourself with many editions of that application. Maintaining and testing these editions can be costly, so it is important to implement much of your application in a single structure of source code that the various editions share.

Generally, the more detailed an application, including the user interface, the more regional variations it will encounter. Streamlining certain operations and processes can help limit the number of international issues that your application must handle. However, streamlining does not mean making your application a generic one that is strongly based on one locale’s practices and requirements. Even a streamlined application should be examined closely to make sure that it does not make worldwide users adapt to the conventions of a single country. Ideally, a localized application appears as though it originated in that locale. If you decide the overhead of creating, testing, and maintaining more complex, localized code is not feasible, at least make sure that your streamlined application does not create usability problems for any locale.

There are instances when the strategy of creating local modules that the shared source code calls is necessary. For example, a real-estate application that manages property-tax information might be structured so that the part of the code that handles taxes is completely modularized, with the legal, business, and cultural issues—tax laws, currency conventions, rounding rules, debit/credit notations, calendar variations—addressed separately for each country. The main procedure conditionally calls the appropriate procedure, such as `ger_tax.p`, `dan_tax.p`, or `no_tax.p`, as the following code shows:

```
CASE CURRENT-LANGUAGE:
  WHEN "German" THEN RUN ger_tax.p.
  WHEN "Danish" THEN RUN dan_tax.p.
  OTHERWISE RUN no_tax.p.
END CASE.
```

The CASE statement uses the value of the CURRENT–LANGUAGE variable, which your application must set, to determine which procedure runs. The procedure `no_tax.p` is a module you use if taxes do not apply. The other modules are fully localized, as they contain the tax laws and financial conventions for Germany and Denmark. By consolidating localization requirements into a few clearly identified modules, the major portion of the application can remain in a single set of source code.

4.1.2 Processing By Characters

To keep your application code flexible so that it can handle multi-byte character data (the Chinese, Japanese, and Korean languages use double-byte code pages), do not process character data byte-by-byte. Also, do not assume that two bytes always equals two characters. By default, Progress functions process character data as whole characters, not as bytes. Make sure that you process all character data at the character level when appropriate. For example, a string-processing routine that examines each byte can mistake the second half of a double-byte character as a new character.

The following Progress code processes data by characters, not bytes, and as a result calculates the storage required incorrectly:

```
DEFINE VARIABLE total AS INTEGER INITIAL 0.
DEFINE VARIABLE floppy-size AS INTEGER INITIAL 1457664.
OUTPUT TO NAMELIST.TXT.
FOR EACH customer WHILE total < floppy-size:
EXPORT name.
/* Calculate file size in bytes by adding
** name length and the 2 quotes and carriage return and linefeed
*/
ASSIGN total = total + LENGTH(name) + 4.
END.

OUTPUT TO terminal.
DISPLAY "NAMELIST.TXT is " total "bytes".
IF total >= floppy-size
    THEN DISPLAY "NAMELIST.TXT will not fit on 1 floppy disk."
```

This procedure exports customer names to a file `NAMELIST.TXT` that is intended to fit on a 3.5-inch, 1.44 MB floppy diskette. The procedure quits if the list of names becomes larger than 1,440,000 bytes. However, since by default, the LENGTH function returns the character count, not the byte count of a string, this procedure does not work properly for multi-byte data. To correct the procedure, use `LENGTH(name,"RAW")` instead of `LENGTH(name)`. Using the LENGTH statement with *type* set to RAW tells Progress to provide the byte count, not the character count.

Another practice that might cause problems for international applications is using a specific numeric value for a character. Different locales use different character sets that have different numeric encoding systems. The letter “é” maps to the hexadecimal value E9 in the ISO 8859–1 code page, but maps to the hexadecimal value 82 in the IBM850 code page.

4.1.3 Using Variables

Another programming convention that helps keep application code flexible is isolating regionally specific information and substituting variables for embedded constants. For example, if an application runs a report with labelled columns for each business day, you can store the column characteristics (labels and position) in variables so that your application can adjust to the fact that the definition of a work week varies worldwide.

Avoid hard coding—that is, embedding explicit strings or constants in a program so that you can only change values by changing the source code. Changing the source code, in turn, requires additional testing of each new version.

4.1.4 Coding With Translation In Mind

Progress includes, as an option, the Translation Management System, a product that supports the translation of Progress applications. The Translation Management System extracts translatable strings from your application code and presents them to the translator. To use the Translation Management System on your code, you must mark untranslatable strings with :U, as the following example shows:

```
IF SESSION:DISPLAY-TYPE = "GUI":U AND VALID-HANDLE(W-Win)
  THEN W-Win:HIDDEN = yes.
```

The :U following the string “GUI” marks the string as untranslatable, and the Translation Management System does not include it in a translation database.

The code that the Progress AppBuilder generates also follows this convention. That is, AppBuilder puts :U after each string not to be translated.

4.2 Input and Output

The design of your user interface must adapt to the internationalization demands of receiving and displaying information in the format appropriate to the end user. This section makes some suggestions for handling user input and output. See Chapter 5, "Preparing the User Interface," for information on the format of data that the application displays for the user.

Keyboards often present internationalization issues, especially when dealing with multi-byte character sets, such as the double-byte character sets required by Chinese, Japanese, and Korean Languages.

The user interface is not the only point where your application presents data to a user. Your application might use printers as output devices. In this case, there are more internationalization issues to consider, such as hardware and software variations and standard paper sizes.

4.2.1 Keyboards

The physical layout of keyboards varies from country to country. In addition, many languages require special key combinations to extend a keyboard's character support. When you design your application, keep in mind that keys might have different assignments. Key mappings that are ergonomic and support good work flow for one keyboard might be quite awkward and unusable for another. See Chapter 10, "Deployment and Configuration," for information on customizing Progress keyboard mapping.

Applications running in countries that use multi-byte character sets often have to rely on an Input Method Editor (IME), which allows users to enter characters that exceed a keyboard's capabilities. For more information on IMEs, see Chapter 8, "Using Multi-byte Code Pages."

4.2.2 Printers

If your application sends output to a printer, you must consider the fonts that printers typically support and the common paper sizes.

By default, when Progress sends output to a printer, it uses the font specified in the `progress.ini` file or Windows registry. If you want more control over the printing environment so that a report's layout is standard, specify the print font by setting the PrinterFont variable in the [Startup] section of the `progress.ini` file.

Progress supports the use of the Windows registry to store and maintain configuration information. Information from the `progress.ini` is added to the registry upon installation. For more information on the `progress.ini` versus the registry and how you can edit configuration data, see the *Progress Client Deployment Guide* and the *Progress Installation and Configuration Guide Version 9 for Windows*.

Paper size is another variable that can affect the quality and usability of your application's output. A report that maximizes the space offered by A4 paper might lose an essential row when printed on an 8.5 x 11-inch sheet. A report designed for 8.5 x 11 paper might lose a column on A4 paper.

4.3 Data-processing Issues

The issues that your application must address when processing culturally-specific data include handling a variety of numeric formats, measurement systems, currencies, date, calendar, and time conventions, and address standards.

4.3.1 Numeric Formats

Numeric data has a cultural component that any international Progress application must address. Numeric separators (decimal and thousands separators) differ across locales. Table 4–1 shows how various countries use periods (.), commas (,), and spaces in numeric values.

Table 4–1: International Numeric Formats

Country	Thousands Separator	Fractional Separator	Example
France	Space	Comma (,)	1 000 000,00
Germany	Period (.)	Comma (,)	1.000.000,00
United Kingdom	Comma (,)	Period (.)	1,000,000.00

Internally, Progress expects all numeric data to follow the United States standard, which uses a period as a decimal separator and a comma as a thousands separator. Your application code must follow these standards when specifying data. However, an application can accept input and display data in the European format if you start the client session with the European Numeric Format (-E) startup parameter.

If your application handles data from various locales, it must have consistent standards for storing and manipulating numeric data. For example, you might want to implement a single application that allows a manager in Rome to enter data using one format while a manager in Tokyo enters data using another format. No matter the format used for entering data, Progress stores DECIMAL data in a neutral format. However, any references to numeric values in code, such as decimal constants or in the Data Dictionary, must use the period as a decimal separator and the comma as a thousands separator, as the following code shows:

```
DEFINE VARIABLE price AS DECIMAL.
DEFINE VARIABLE price1 AS DECIMAL.
. . .
price1 = price * .10.
DISPLAY price1.
```

When you run this code with the -E startup parameter, the value of price1 displays with a comma as the decimal separator.

Within an application, you can also use the NUMERIC-FORMAT attribute of the SESSION handle. It indicates whether periods and commas follow the American convention or the European convention. The American convention uses commas to indicate thousands and periods to indicate decimals. The European convention uses periods to indicate thousands and commas to indicate decimals.

4.3.2 Measurements

Measurements are a type of numeric data with several cultural components that your application must handle. An international application must not hard code any manipulations of measurement data. When working with measurements there are two aspects to consider: nonmetric units of measurement and industrial standards.

The international system for spatial measurement is the metric system. The most common system for measuring temperature is the Celsius system. If one of the locales where your application runs uses a different measurement system, the application should convert data so that your users can work with the measurement system they expect.

In most business or industrial applications, a simple conversion of kilograms to ounces does not allow users to work with the measurements they know best. A simple conversion often results in measurements that are awkward or that do not reflect local standards. For example, a standard size for photographs is 10x15 centimeters. In the United States, the equivalent standard is 5x7 inches. Converting from metric or from inches gives you accurate equivalencies, but not useful ones. The application designer must analyze the measurement systems and standards used by the target business sector in the various locales. In some cases, your code might use a straightforward conversion. In other cases, your code must work with equivalencies.

You should create an internal standard for handling measured data to avoid inconsistencies. A suggested technique is to print, display, and input measured data in the format needed by users and convert it to the appropriate internal standard data format for storage. If you use the same standard representation throughout your application, you have to write only two conversion routines for each format: one to convert from input format to the internal standard and one to convert the internal standard to its display format. You can perform all internal calculations referencing one format and avoid making changes throughout your application during localization. You can also use the same data for all locales; all you have to do is update a conversion routine to reflect a new display format.

4.3.3 Currencies

If you are designing a single application that receives currency data from different countries, you will have to develop a standardized mechanism for handling conversions. When your application manipulates currency data, it must handle:

- Currency symbols—length and placement
- Decimal and thousands separators
- Notation for negative numbers

Do not hard code your application to expect currency symbols in a specific position, as the length and position of currency symbols vary. When displaying currency, do not allow only one or two spaces for the currency symbol. Some currency symbols are six characters long, such as CFA Fr used by a group of African nations. Also, remember that some currencies commonly have values in the millions (such as the Italian Lire) and require more display space.

For an application where the user works with various currencies, consider using combinations of widgets or parameters that allow the user to specify a currency. For example, use a fill-in field with a combo box to allow the user to fill in an amount and select the currency type. Separating the currency data into two components—amount and type—makes it simpler to process.

The notation for negative numbers also varies from locale to locale. Negative numbers can be indicated by a minus sign (-50), parentheses ((50)), or an abbreviation, such as DB (DB50). Your application should be able to process negative numbers, regardless of the format that users follow for input.

4.3.4 Dates

There are several cultural components affecting dates that your application must consider when processing input:

- Calendar type
- Date format
- Business week

There are several different calendars (Japanese Emperor, Buddhist, Gregorian) in use around the world. Some countries use more than one calendar at a time, depending on the context. For example, the Japanese business community uses the Gregorian calendar, but the official calendar is the Japanese Emperor calendar, which has as its year one the first year of the current Emperor's rule. Also, there are many different formats for indicating dates within each calendar.

The difference that might have the most far-reaching impact on your application is how calendar dates are presented in different countries. The same date from the Gregorian calendar could be indicated as 16-2-95 in Ghana, 16.2.1995 in Germany, and 2/16/95 in the United States. The day/month order is different, as are the separators. Some locales use character symbols for the month or write it out. Note that your application must be sensitive to differences in dates not only in the case of user data, but also when it receives date information from the operating system, which is configured to support local conventions.

Another difference that might affect your application is the cultural definition of a week. The week's starting point (Sunday or Monday, for example) varies widely. In France, the first day of the week is Monday; in the United States, it is Sunday. Hard coding an extensive database backup routine that takes place on the evening of the fifth day of the week works well in France (day 5 is Friday), but might not be appreciated in New York where day 5 is Thursday and still the middle of the work week. Also, if you are writing an application that deals with the business week, consider that not every locale divides the week into the same five workdays and two leisure days. Arab countries do not typically conduct business on Friday. Some countries do not close businesses for a two-day weekend.

If you hard code date formats and dates, your application will have to be rewritten for any locale that expects a different calendar, date format, or business week.

Handle dates using the DATE data type. Store date data as integer data which requires less storage space and is accessed more quickly than character data. Also, integer data, unlike character data, is independent and does not have to be converted for each market because of variations in characters sets or date formats. Whenever you include a date in a procedure or specify a date in the Data Dictionary, you must use the integer data type.

Also, Progress provides the following methods for you to specify the display format of a date:

- The FORMAT option of the Format phrase allows you to specify a display format for all DATE data.
- The Date Format (-d) startup parameter allows you to specify the order to display date information during a Progress session. The internal storage format remains the same. The -d startup parameter is client startup parameter only.
- The DATE–FORMAT SESSION handle attribute allows you to specify the format for displaying dates during the current Progress session.
- The Century (-yy) startup parameter allows you to specify the current start date for the Progress two-digit year range of 100 years. See its reference entry in the *Progress Startup Command and Parameter Reference*.
- The DATE, DAY, MONTH, and YEAR functions allow you to convert data (string and numeric) into the DATE data type.

For more information on the startup parameters, see Appendix A, "Progress Resources," and the *Progress Startup Command and Parameter Reference*. For more information on the functions, see the *Progress Language Reference*. For an example of how to handle dates, see the following procedure (p-snhndl.p).

p-snhndl.p

```
DEFINE VARIABLE mynum AS DECIMAL.

SESSION:DATE-FORMAT = "mdy".
DISPLAY SESSION:DATE-FORMAT SPACE(5) TODAY WITH FRAME d1.

SESSION:DATE-FORMAT = "dmy".
DISPLAY SESSION:DATE-FORMAT SPACE(5) TODAY FORMAT "99-99-99"
  WITH FRAME d2.

DISPLAY SESSION:DATE-FORMAT SPACE(5) TODAY FORMAT "99.99.99"
  WITH FRAME d3.

mynum = 12345.67.
SESSION:NUMERIC-FORMAT = "AMERICAN".
DISPLAY SESSION:NUMERIC-FORMAT mynum WITH FRAME f1 NO-LABELS.

SESSION:NUMERIC-FORMAT = "EUROPEAN".
DISPLAY SESSION:NUMERIC-FORMAT mynum WITH FRAME f2 NO-LABELS.
```

The p-snhndl.p procedure illustrates the read and write capability of the DATE–FORMAT and NUMERIC–FORMAT SESSION handle attributes. Note that Progress always handles data using periods as decimal separators within source code, even when you specify the EUROPEAN display format. The display format affects input and output when the application is executed, not when it is compiled.

4.3.5 Time

Time formats differ around the world because some countries use the twenty-four-hour clock and others use the twelve-hour clock. Some countries use both, depending on the business sector. The format for representing clock time varies also due to the separators between hours, minutes, and seconds. For example, the same time of day can be displayed as 2:42:10 PM in Canada, 14.42.10 in Finland, and 14h42:10 in South Africa. To accommodate these differences, you can use techniques similar to those used for dates. Progress represents time as a fraction of a day. For example, one half of a day is 12:00 noon, and one quarter of a day is 6:00 AM.

4.3.6 Addresses

There are many styles of address and phone number formats. Avoid hard coding processing routines for addresses or phone numbers that can change when your application is used in another country or language. For example, postal code formats are very different in various countries. Addresses are not always written with the street address immediately after the name, and the number of lines or information fields varies. The size of information fields also varies. The United States uses a nine-digit postal code while Great Britain uses a six- or seven-character combination of numbers and letters. Names can also require different ordering in countries where the first word in a name is the family name. Some locales use a single title when addressing people while others list all the titles an individual can claim. The position of a title varies also. In Japan, the title follows the name. In the United States, one title can precede a name while another follows it (for example, Mr. Eric Henderson, Jr.).

Postal codes deserve attention as they deviate widely. Storing postal code data as CHARACTER data instead of INTEGER data gives your application more flexibility in handling a wide variety of alphanumeric formats. Make sure that any routines that parse addresses can handle international data. If your application runs internationally, that is, with users entering data at many locations, use validation routines to check that address entries have the correct format.

4.4 Sorting Data

Similarly, any comparisons of characters that rely on the value a character has in a given code page result in code that returns false results. The value a character has in a code page is not necessarily its sort weight. For example, using the CHR function, which converts a numeric value into a character, can produce errors. The condition, IF Z < CHR(255), could be false if the character with the code-page value of 255 has a sort weight of 1. In this case, no character can come before the character that sorts as 1.

4.4.1 Local Conventions

Ordering of character data is regulated by language rules and cultural conventions. For example, Japanese scholars have one system for ordering and classifying Kanji characters, but in a business context (a Tokyo telephone directory, for example) other conventions apply. European languages and countries present some of the same issues. Be sure to research the ordering conventions that affect CHARACTER data in the locales where you plan to run your application.

Consider German, which uses many of the 256 characters present in a common European character set. The base alphabet (A to Z) is extended by the Umlaut (Ä, Ö, Ü) and ß. Truly localized applications for German-speaking countries cannot default to sorting these characters based on their value in a code page. Each country has its own conventions for ordering these special characters. A dictionary published in Germany considers A and Ä to be interchangeable for ordering purposes. A Swiss-German dictionary, however, lists all words beginning with A before all words beginning with Ä.

You can control the order in which Progress sorts and compares character data by using collation tables. A collation table assigns values to characters; Progress uses these values to weight a character in a sort order. For example, code page ISO8859–1 assigns the Ä the value 196 and assigns A the value 65. If character data were sorted according to the values in the code page, all customers whose names begin with Ä would appear after those whose names begin with Z, which is not the order an analyst in Frankfurt expects. In Germany, A and Ä are commonly sorted together. A collation table for Germany would assign 65 and 196 proximate sort weights.

Your application might require converting all character data to uppercase or lowercase before it is processed or stored. When the Progress CAPS and LC functions perform these case conversions, Progress uses a case table to match a character to its uppercase or lowercase equivalent.

4.4.2 Using Collation Tables

Each database has its own collation tables that Progress uses for index operations. The 4GL also uses collation tables for string comparisons. The collation tables specify the order in which characters sort. These tables are named CASE-INSENSITIVE-SORT and CASE-SENSITIVE-SORT. The CASE-INSENSITIVE-SORT table sorts uppercase and lowercase letters identically. The CASE-SENSITIVE-SORT table distinguishes between uppercase and lowercase letters. To control how Progress sorts characters, you can modify these tables. Each collation table consists of 256 cells (16 rows of 16 cells), each of which contains a decimal numeric value.

For each character in a string, Progress indexes the character's numeric value from the collation table to a cell in one of these tables. For example, if a character has a numeric value of 233, Progress goes to cell 233 in the table. After Progress locates the appropriate cell, it reads the value in the cell to find the character's sort weight. The *sort weight* tells Progress where you want the character sorted relative to other characters in the code page. For example, a sort weight of 001 means that the character sorts first in an ascending sort.

Collation tables must exist for each code page. You can have more than one collation per code page. To find the correct collation table, Progress uses the values of the -cpinternal and -cpcoll startup parameters as keys to the `convmap.cp` file. The same collation name can appear for multiple code pages.

The following rules determine how Progress decides which collation table to use when performing comparisons or sorting:

- If you specify collation tables with the Collation Table (-cpcoll) startup parameter, Progress uses those collation tables.
- If you name databases in the startup or connection command and do not use -cpcoll, Progress uses the collation table defined for the first database. If you connect to an additional database, Progress uses the collation table of the additional database when working with the additional database.
- If you do not name databases in the startup or connection command and do not use -cpcoll, Progress searches `convmap.cp` to find the collation table named BASIC for the internal code page.

NOTE: The collation table you specify with the -cpcoll startup parameter must be in the `DLC/convmap.cp` file, which is a binary file that contains tables for managing characters created by compiling the `DLC/prolang/convmap/convmap.dat` file. If `convmap.dat` does not have a collation table you need, you can create your own.

4GL Comparisons

Progress provides two types of character sorting. One type governs how data is stored within a database and affects index operations. Another type affects *4GL comparisons*—comparisons that are performed by the 4GL and that do not impact indexes. For example, this is a typical 4GL comparison:

```
IF character-expression1 > character-expression2
```

The character expressions can contain character strings, character variables, or character fields.

Database Collation

A database has a collation table stored as part of its schema information. The collation table that is stored in the database does not affect how the Progress client or server compares or sorts CHARACTER data (except for indexed data); it controls only how indexes are sorted and assigned. Similarly, a collation table specified by -cpcoll does not affect the internal collation order that a database uses for its indexes.

Progress supplies collation tables for databases for many languages and locales. These are located in the `DLC/prolang` subdirectories. For languages that have one commonly used collation table, the table is contained in the `_tran.df` file in the language subdirectory. If more than one collation table is common, each is contained in a data definition (`.df`) file named for the table's corresponding code page. To change the collation table associated with a database, load the appropriate data definition file and rebuild your indexes.

For more information on collation tables, see Chapter 3, "Understanding Character Processing Tables."

4.4.3 Using Case Tables

Developers design applications to support different market requirements, for example, different character sets and different sort order of characters within a character set. The rules for changing case might also be different, and our applications must allow for this. For example, Japanese, Chinese, Korean, Hebrew, Arabic, and Thai alphabets have only one case and do not have uppercase and lowercase versions of letters. Other cultures do have different case rules. In France, lowercase letters with accents, for example, the character "é," lose the accent when capitalized. Therefore, the character "é" becomes "E." However, in Canada the character "é" retains the accent and becomes "É" when capitalized.

Progress provides many case tables to accommodate different case rules for different markets. You can also create your own case rules by creating your own case tables. When you deploy your application, select the appropriate case rules by using the Case Table (-cpcase) startup parameter. The -cpcase startup parameter affects the CAPS function, the LC function, and the FORMAT character (!). The CAPS function specifies uppercase rules, and the LC function specifies lowercase rules. The FORMAT character (!) is used to specify that data must be uppercase. The case table defines the uppercase and lowercase equivalents for each character in the code page. For example, the French case table defines the uppercase equivalent for the character “é” as “E.” If you apply the LC function to the lowercase “é,” it does not change. The French-Canadian case table however, defines the uppercase “é” as “É.”

These differences in case rules are important because an application that uses the wrong case rules might change the meanings of words. Case rules can also affect your queries. For example, using French case rules you cannot find the name “René” with the following query:

```
FOR EACH customer WHERE CAPS (name) MATCHES "RENÉ":
```

The query does find the name “René” using French-Canadian case rules because the French case table defines the uppercase equivalent of “é” as “E,” not “É.”

In France, the query must run with an uppercase “E” and no accent as follows:

```
FOR EACH customer WHERE CAPS (name) MATCHES "RENE":
```

This query finds the name “René” only if you use the French case rules, not the French-Canadian case rules.

Progress has a default case table for every code page that it supports. The case tables are in the `DLC/prolang/convmap.dat` file. For more information on the `convmap.dat` file, see Chapter 2, “Understanding Code Pages.” The default case table for a given code page is called BASIC.

Your application might require a different mapping between uppercase and lowercase characters than the one that the BASIC case table provides. Progress allows you to specify which case table you want to use at startup. You can use the Case Table (-cpcase) startup parameter when you start a Progress client session or server.

The following rules determine how Progress decides which case table to use for case conversions:

- If you specify a case table with the (-cpcase) startup parameter, Progress uses that case table.
- If you do not use -cpcase, Progress searches `convmap.cp` to find the case table named BASIC for the internal code page.

This is a sample startup command that sets the code page and case table:

```
pro parisdb -cpinternal ibm850 -cpcase French
```

This command uses -cpcase to specify that the Progress client use the French case table when performing case conversions instead of the BASIC case table for the IBM850 code page.

NOTE: The case table you specify must appear in the `convmap.cp` file, which is a binary file that contains tables for managing characters created by compiling the `DLC/prolang/convmap.dat` file. If `convmap.dat` does not have a certain case table, you can create your own.

For more information on case tables, see Chapter 3, "Understanding Character Processing Tables."

4.5 Compiling Translated Applications

If you use the Progress Translation Manager System to translate your application, you must connect to the appropriate translation databases in addition to any application databases. You can compile all the language editions into a single r-code file by using the LANGUAGES option of the COMPILE statement. This code example specifies the languages to be read from the appropriate translation database or databases. The translated character strings are stored in segments within the r-code:

```
COMPILE myfile.p LANGUAGES(Dutch,English,American:English,French,German).
```

All of the languages in the example list above can use the same code page, ISO8859–1. If the list included Russian, you would have to run this statement again in a separate session with -cpinternal set to a code page that supports Russian so that the r-code would have a valid code page.

If you compile r-code with -cpinternal set to one code page and you run the r-code in a session with -cpinternal set to another code page, the two code pages must be compatible. Progress must be able to convert one code page to the other. There are RCODE–INFO handle attributes that allow you to read information about your r-code and about a session to determine if your code pages are compatible. Table 4–2 lists the RCODE–INFO handle attributes that apply to international or localized applications. See the COMPILE statement and RCODE–INFO handle entries in the *Progress Language Reference* for more information.

Table 4–2: RCODE–INFO Handle Attributes

Attribute	Description
CODEPAGE	Indicates the code page of text strings stored in the text segment of the r-code.
LANGUAGES	Holds a comma-delimited list of the languages supported by the r-code.

Because r-code compiles with the code page set by -cpinternal, you must set the R-code Out Code Page (-cprcodeout) startup parameter to compile code with another code page. For more information on the -cprcodeout startup parameter, see the *Progress Startup Command and Parameter Reference*.

5

Preparing the User Interface

The design of the user interface is crucial to the success of any application with a worldwide audience. If you want to be competitive in a global market, you must design a user interface that meets the needs of each audience.

Try to design a user interface that is:

- Generic enough to be used as is or easily modified for any audience.
- Easy to learn and use. A user interface that is customized for the audience gives the user more confidence in the product and better suits the user's needs.

You also want to create a user interface that works well for your users. If you design with internationalization in mind, the user interface will support many languages. In addition, you will not have to increase development efforts, redesign for every audience, or schedule extra time for maintenance and quality assurance.

This chapter contains the following sections:

- Screen Layout and Composition
- Graphics and Icons
- Designing With the AppBuilder
- Designing To Allow For Translation

5.1 Screen Layout and Composition

The layout of the user interface should be logical and easy to use. Menus and graphics that are used frequently should be easy to find and easy to recognize.

If the user interfaces you design can share one design for most or all markets, there is less maintenance to perform and more consistency. Remember that you might need to make changes to the user interface because of regional equipment differences or cultural requirements.

5.1.1 Equipment Differences

In some cases, the layout and composition of the user interface must change from one language edition to another because of equipment differences. Computer companies often manufacture different models for each market. Technology standards vary from one country to another. Differences in keyboards and monitors might affect the design of the user interface. Recognize that many users do not have access to the latest computer components that might be available elsewhere. For example, some markets might not consistently have access to color systems, advanced graphics, large memory systems, or high resolution printers.

Monitors

There are many different models of monitors available, with features or limitations that might affect the look of the user interface. Consider this in the design phase. In general, you should design for a range of monitors and not just for the ones that offer the newest features. Monitor features that might affect user interface design include:

- Color

  Monochrome monitors are available in black and white, green and black, or amber and black. Gray-scale monitors are monochrome monitors that display different shades of gray. Color monitors vary in capability (depending on the model and video card) and can display 16 colors, 256 colors, or even millions of colors.

  Choose a color scheme that is available and looks acceptable on all of the monitors you support. For example, choose a scheme that looks acceptable on both a monochrome monitor and a color monitor. Make color selection user-definable so the user can select colors based on what is available.

- Resolution

  Refers to the number of dots (pixels) on the screen. The higher the number of pixels the sharper the image. A common resolution is 640 dots on each of 480 lines. However, the Japanese NEC PC monitor displays 640 dots on each of 400 lines. Therefore, you lose a couple of rows of text on this monitor. If you design a screen that uses every row on a VGA 640 by 480 display, it cannot be used on these Japanese PCs. View your design before you finish it to see if the resolution on most screens supports your design.

- Screen Size

  The amount of viewing space available on each monitor differs from model to model because of differences in monitor dimensions. Screen size is measured diagonally from one corner of the screen to the other. Some screens are smaller or larger than others. Additionally, screen orientation can be either portrait (height greater than width) or landscape (width greater than height). The design of the user interface must allow for these differences in screen size and orientation.

Keyboards

Keyboard layouts vary to support different languages so it is important to consider the variety of keyboards during the design phase. For example, you should not hard code short-cut keys (key combinations that perform specific commands) until you ensure that the short-cut keys are available to your audience. In addition to the availability of keys, there are ergonomic issues to consider. The short-cut keys you select should be easy to type. For example, typing Ctrl-} on a United States keyboard is simple, however on a Finnish keyboard it requires three separate keys to complete the same function. Also, the at symbol (@) is one keystroke on United States keyboards but three keystrokes in some markets.

5.1.2 Culturally-specific Issues

You cannot rely on translation to make the user interface audience specific. You might have to design a user interface for each audience to accommodate the local conventions. For example, you might see the following translation of the label for a phone number field:

English	French
Telephone	téléphone

However, a more appropriate label for this field is “Numéro de téléphone.” The first translation is correct but not as appropriate as the second translation if you consider the context.

Also, you cannot assume that the translation into a particular language allows for all of the dialects of that language. For example, translation into Spanish often does not account for regional differences between various Spanish-speaking users. Progress provides three Spanish-language translations of `promsgs`: Castilian, Mexican, and South American to meet local language requirements. If you design separate language editions you can create fields that are language specific.

You might also have to design different user interfaces or interface objects to accommodate local conventions. Local conventions are discussed in the following sections:

- Abbreviations and Acronyms
- Address Formats
- Calendar, Date, and Time Formats
- Colors and Sounds
- Numeric Formats
- Currency Formats
- Field Labels and Field Sizes
- Financial Rules
- Names and Titles
- Phone Number Formats

Local conventions might require you to design separate user interfaces for different audiences even if the audiences speak the same language. For example, you might design a user interface that has a “County” field for an Irish market and another user interface that has a “Province” field for a Canadian market.

You might also need to consider field size. If the user interface includes a field that allows a two-character entry for the state, a county or province name does not fit. So, you cannot simply create a field with the label “County/State.” The field must be large enough to accommodate the character input.

Abbreviations and Acronyms

Abbreviations (shortened forms of words or phrases) and acronyms (words created from the initial letters of words or phrases) used in one locale are not always understood in other locales. For example, the English ordinal abbreviations “1st” for first, “2nd” for second, etc., have no equivalent in modern Hebrew. In France, there is an organization that is known by two different abbreviations, ISDN (Integrated Standard for Digital Networks) and RNIS (Réseau Numérique Intégration de Services). The abbreviation for the International Organization for Standardization (ISO) also causes confusion. People often assume that ISO is an acronym for International Standards Organization.

Sometimes acronyms are well known only to some groups within a country, for example, people in the military or certain industries. You might see a sign near industrial areas with the acronym “Hazmats” for “hazardous materials.” Not everyone will understand this acronym. Also, some cultures do not use abbreviations, for example, China.

Address Formats

Address formats—address fields, address-field lengths, and address-field order—vary from country to country. In Japan, addresses are written in order by country, city, street, then addressee. In Ireland and the United Kingdom, addresses include the county and are in order by addressee, street, city, county, then country.

Address formats might also include miscellaneous information like building names or names of regions within a city. The number of fields and the number of characters within each field must be considered. A small address format in one language edition might require more space in another language edition. The use of titles to address someone varies also.

Calendar, Date, and Time Formats

Different cultures follow different calendars. The week might run from Monday to Sunday or Sunday to Saturday depending on the culture. Different cultures might also follow different calendar years. For example, the year 1996 is 2539 on the Buddhist calendar and Heisei 8 on the Japanese Emperor calendar.

Date and time formats also vary from culture to culture. Table 5–1 illustrates just a couple of the many date and time formats you might see.

Table 5–1: Date and Time Formats

Country	Date Format	Time Format
France	18 septembre 1996 18.9.96	16.45
Hungary	1996. oktober 18. 1996. 10. 18.	16:45
United States	18 September 1996 September 18, 1996 9/18/96	4:45 PM 16:45

Punctuation and capitalization in the date and time format also vary. A slash (/) might separate the numbers in a date format (as in 1996/9/18) or a period (.) as in (1996.5.4). You must design date and time formats that are appropriate for the audience.

Colors and Sounds

How you use colors in your interface can help or hinder its usability. Associations or inferences made with colors might not be understood worldwide. For example, the color red means very different things throughout the world. Red signifies happiness in China, loyalty in France, and danger in the United States. Do not use colors in a way that could diminish the user's understanding of your application. If you imply meanings with colors and you do not modify the colors you use for each country, you might alienate some of your users.

The meanings of sounds are subjective and vary from culture to culture. For instance, telephones have different rings in different countries. So do sirens. If you use a sound as part of your software, it might not be recognized in every market and it might be misinterpreted.

Sound should accompany other types of communication, not replace them. For instance, a beep might accompany an error message, but the error message serves as the primary indicator that something is wrong. Sound communication should not be used alone because sound is not available to all users for a variety of reasons such as:

- Sound equipment is not available for the market or the individual
- Sound equipment standards vary from culture to culture
- Sound cannot be heard in a noisy environment or by the hearing impaired
- Sound is user controlled and can be shut off.
- In some cultures it is not desirable to have the machine beep when you make a mistake. The beep might cause embarrassment in an open office environment.

Numeric Formats

The symbol that represents the fractional separator (the symbol that separates the fractional portion of a number from the integer portion) and the thousands separator (the symbol that separates each group of three digits in a number), can vary greatly between locales.

Currency Formats

The symbol that represents the type of currency and the general format can vary greatly between locales. For example, positive and negative currency values often have different formats. You must consider these issues when you create currency fields and field labels. Table 5–2 shows examples of different currency formats.

Table 5–2: Currency Formats

Country	Currency Value (Positive)	Currency Value (Negative)
Germany	1022,89 DM	- 1022,89 DM
Italy	L. 1022	-L. 1022
Norway	kr1022,89	kr-1022,89
Portugal	1022$89 Esc.	-1022$89 Esc.
United Kingdom	£1022.89	-£1022.89
United States	$1022.89	-$1022.89

NOTE: Although the preceding formats show the same numerical amount, the actual amounts vary based on current exchange rates.

Field Labels and Field Sizes

Consider field label names and field sizes during the design phase. For example, an application written for a user in the United States might have a user-defined field that reads “Social Security Number.” This field might have to read “Government ID Number,” or something else in another country. The field size might also change if the value is different, for example a nine-digit value instead of a ten-digit value.

Financial Rules

The rules for rounding numbers vary from country to country. The following are some of the methods for rounding numbers:

- Round numbers up or down using .5 as the determining value. For example, round numbers in the range 2.1 and 2.4 down to 2.0, and numbers in the range 2.5–2.9 up to 3.0.
- Round numbers up or down using .25 as the determining value. For example, round numbers in the range 2.0–2.25 down to 2.0, and numbers in the range 2.26–3.0 up to 3.0.

Names and Titles

Different cultures have different rules of social etiquette for addressing or referring to a person. German business people often use the titles “Herr,” as in “Herr Direcktor” and “Frau,” as in “Frau Direktor.” The Japanese append suffixes to names. For example, “Michael-san” is a polite way to refer to someone named “Michael,” as in “Mr. Michael” or “Herr Michael.”

The use of first, middle, and last names varies from culture to culture as well as the order in which the names appear. For example, names in Icelandic phone books appear in order by first names then last. In Indonesia, people are known by only one name.

You must allow for differences in naming conventions. Do not assume the order of names or titles.

Phone Number Formats

Phone number formats vary from country to country. In the United States, phone numbers are ten digits, including the area code. In Norway phone numbers are eight digits; at one time there was a two-digit area code for metropolitan areas, but this has been discontinued. Also, the layout and punctuation of phone numbers change from country to country. The following examples show the same phone number in a variety of formats:

(64) 12512

(064) 12512

64-12512

Each of the examples shown above is an acceptable phone number format. Design phone number fields to accommodate appropriate formats.

Language Issues

The expansion of the worldwide market necessitates the availability of software applications in many language. A variety of issues arise from the need for applications that work well in many languages. The following are four of the many language issues:

- Direction that text is read

  Some languages are read from left to right, and some from right to left. This can influence your text design. Users who read right to left look for the most important information in the upper-right corner, while users who read left to right look for the most important information in the upper-left corner, so you must change the user interface accordingly. For example, order tab folders in the direction each is read, so that the most important or the most frequently used tab folder displays first.

- Input Method Editors (IME)

  Some cultures require a character input method editor on the screen. For example, Asian markets require an IME. A character input method editor is a system for generating characters through a sequence of keystrokes. The IME requires a lot of screen space and decreases the amount of available display space for the user interface. Design your interface with extra room to allow for these differences.

- Letters and characters

  Language differences often necessitate the use of different letters and characters on the user interface. Different fonts and varying type-size might be available. Keyboards are available in a variety of key combinations that are designed specifically for certain languages.

- Symbols and abbreviations

  If you use symbols or abbreviations to replace text or to save space on the user interface, ensure that the symbols are available on the keyboard and that the abbreviations are appropriate. For example, in some languages the abbreviation might be longer than the original name or does not exist at all. Short-cut keys might need to change in order for the mnemonic abbreviation to have meaning.

5.2 Graphics and Icons

Many organizations develop regional or international standards in computer graphics. The development of useful symbols in the computer industry is not an exact science. Standards are constantly changing and are not always universal. For example, the Apple trash can icon is not considered an effective “delete file” icon because trash cans of this type do not exist in every country. The meaning of the icon is not clear in every market.

5.2.1 Text In Graphics and Icons

Graphics and icons often include text. Only text that will not be translated (such as a company logo) should be included on graphics and icons that are going to be used internationally because it is very expensive to translate text within a graphic. Whenever possible, avoid text in graphics and icons completely. ToolTips are a good alternative that are less costly to translate.

5.2.2 Images To Avoid In Graphics and Icons

To design graphics and icons that are accepted worldwide, be cautions of the items mentioned in the following sections:

- Images Of Animals
- Images Based On Puns Or Word Play
- Images Of Body Parts Or Gestures
- Images Biased Towards a Particular Culture Or Region
- References To Sports
- Symbols Related To Religion Or Culture

Images Of Animals

Different locales often attribute different attributes to animals. In the West, the pig is considered dirty and lazy, but in China, unintelligent and easily cheated. In Taiwan, the owl is considered brutal, but in the United States, wise. Using images of pigs, owls, and other animals in applications to be deployed in multiple locales might confuse users.

Images Based On Puns Or Word Play

Do not create graphics that are based on puns or word play. For example, using the symbol of a scale for “change size proportionally.” The word for scale and the word for “changing size proportionally” might not have the same relationship in another language. In the United Kingdom, for example, a scale is called a “balance.” Do not assume that any phrase is universal. In the United States it is clear that an “Exit” sign is directing you “out,” but in Taiwan or Ireland this sign might read “Way Out.”

Images Of Body Parts Or Gestures

Some cultures consider certain body parts or gestures offensive. For example, in the United States it is rude to point with the middle finger. In the Middle East it is rude to point with the index finger. In some cultures it is rude to extend a thumb. For this reason, flight attendants point with an index finger and middle finger together. Some cultures consider the sole of a shoe, an elbow, or even the palm of the hand offensive.

Considering the number of countries, villages, and neighborhoods in the world, each with its own social etiquette, it is safer to avoid these images altogether.

Images Biased Towards a Particular Culture Or Region

Do not use images biased towards any one culture or region. For example, should the image of a world map have North America at the center or Asia? It might be a better idea to use an image that does not place one particular country or continent at the center.

References To Sports

References to sports are misleading and confusing. Some sports are only known regionally. For example, baseball, cricket, and sumo wrestling are known in specific geographic areas. A reference to “home run” to imply success is clearly culturally biased. Users from countries that do not play baseball would not understand this reference.

Symbols Related To Religion Or Culture

Do not use symbols related to religion or culture. They might be misunderstood or even give offense. For example, in some cultures birds symbolize bad luck, and in other cultures a skull and crossbones represents danger.

You should also consider the significance of certain numbers. For example, in some countries the number 13 is considered unlucky. Buildings often do not have a floor 13. The number 86 is often used to imply removing or stopping something as in “86 it.” In parts of Asia the number 4 is pronounced the same as the word for death and is considered morbid. Not all numbers have a negative connotation. Some people consider the number 7 to be lucky.

Unless you have done an extensive study of numbers and their meanings in many cultures, you should avoid this type of reference to numbers or symbols.

5.3 Designing With the AppBuilder

The international issues that affect the design of any user interface also apply to working with AppBuilder and SmartObjects. AppBuilder provides a couple of options for designing a user interface that works worldwide. Try to design SmartObjects that are generic enough to work for any audience. Language issues and cultural conventions must be considered. If you cannot use a generic user interface, AppBuilder is a useful tool for customizing your applications for each region (localizing). You can customize SmartObjects by:

- Localizing SmartObject Masters
- Localizing SmartObject Instances
- Using Multiple Layouts

5.3.1 Localizing SmartObject Masters

You can create localized SmartObjects masters. For example, you could create one SmartViewer for Irish addresses and another SmartViewer for United States addresses, each with the appropriate field labels and field sizes for the locale.

Figure 5–1 illustrates a SmartFolder with a SmartViewer designed to display Irish customer addresses. The SmartViewer includes a county field. The Irish address format includes the county and the United States address format does not. You could create a separate SmartViewer for displaying United States addresses.

The size of the telephone number field might also change to accommodate the difference in telephone numbers from location to location. You must consider all audiences within a country as well. The postal code is not necessary in a smaller Irish town like Killarney but in a city like Dublin it is, so the Postal-Code field was added. You must design objects so that all of the necessary information is included. Figure 5–1 shows a SmartObject with an Irish address.

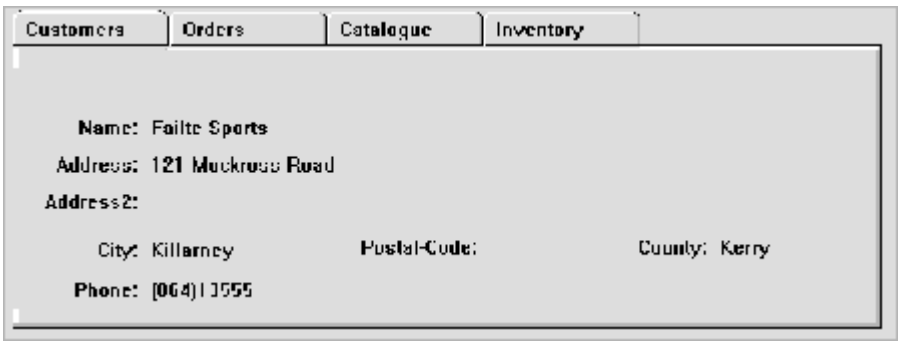

Figure 5–1: SmartObject Labels

5.3.2 Localizing SmartObject Instances

An option to creating localized SmartObject masters is to customize a running instance of a generic SmartObject. SmartObjects have an Instance Attribute dialog box that allows you to specify attributes in a run-time instance of the object without modifying the master object itself. Some of these attributes allow you to meet local requirements that translation does not address, without having to create a new master object for every location. For example, you might create one SmartFolder with German labels and another customer folder with Spanish labels. Figure 5–2 illustrates how you can change SmartFolder tab labels in the Instance Attributes dialog box to create a customized running instance of a SmartFolder:

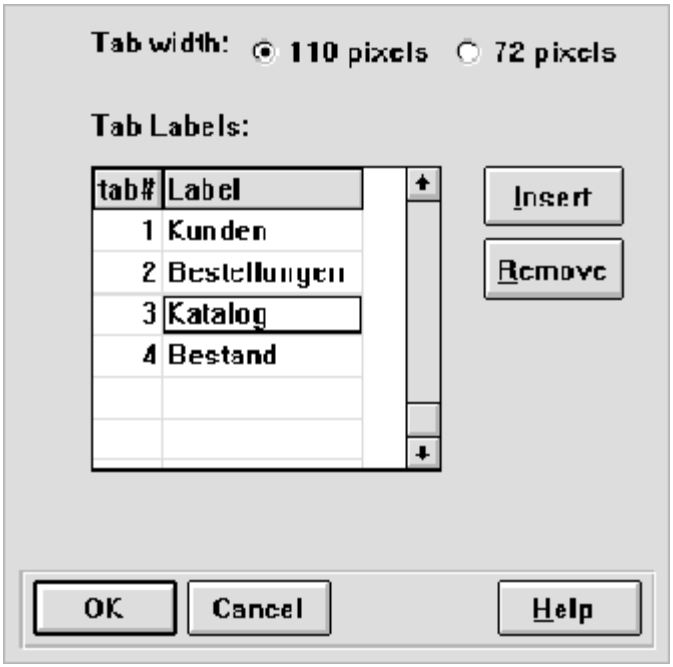

Figure 5–2: SmartFolder Attributes Dialog Box

Figure 5–3 shows a customized running instance of a SmartFolder.



Figure 5–3: Localized SmartFolder

You can localize SmartPanel with the SmartPanel's Attributes dialog box. For example, a graphical Navigation SmartPanel has buttons that represent the functions; First, Next, Previous, and Last. Figure 5–4 shows a graphical Navigation SmartPanel.



Figure 5–4: Graphical Navigation SmartPanel

A user who reads text from right to left expects the button on the far right to execute the “First” function, then “Next”, “Previous”, and “Last” (“Last” being the button on the far left). You can change the direction in which the buttons display from the Instance Attribute dialog box. If a user reads text from right to left, you can make the navigation buttons display from right to left. Figure 5–5 shows the Navigation SmartPanel Attributes dialog box.



Figure 5–5: Navigation SmartPanel Attributes Dialog Box

5.3.3 Using Multiple Layouts

A layout is a collection of widgets and associated attribute settings. These attribute settings determine how the layout appears when you run the application. Each layout has a name and, depending on the type of layout, an associated run-time expression.

At design time, a layout displays in the design window. When you switch between layouts in the design window, the AppBuilder alters the appearance of the design window to match the characteristics of the new layout.

You can also use AppBuilder to support multiple layouts in a single procedure file. The advantage of this approach is that you continue to maintain and deploy only one set of source files for an application. When you run a procedure file that contains multiple layouts, the procedure file adjusts its interface to suit the current run-time environment. You can also easily switch between layouts at run time.

For example, you might have an application that produces address labels. However, address formats vary from region to region. So, you should have a different layout for each address format.

For more information on multiple layouts, see the *Progress AppBuilder Developer's Guide*.

Character-client Layouts

Often a DOS character client and the corresponding Windows client use different code pages. When you run the character client, you must ensure that the AppBuilder Character Run window uses the correct DOS code page. To ensure this, follow these steps:

1 ♦ Create a parameter file (`.pf` file) for the character client. Make sure that -cpinternal and -cpstream are set to the appropriate code page for DOS.

2 ♦ Specify the pathname of the Progress character client startup file with the `PROSTARTUP` environment variable in the [WinChar Startup] section of the `progress.ini` file or Registry.

```
PROSTARTUP = [PATH]/(charstartup.pf)
```

NOTE: The Character Run window does not support the display of multi-byte character sets. If your character application uses multi-byte characters and you run the application in the Character Run window, the character session is displayed incorrectly. Also, Progress error messages might be displayed incorrectly if the character session uses a multi-byte PROMSGS file.

5.4 Designing To Allow For Translation

To create a product for many languages, developers must design and create applications with translation in mind. For example, the developer must create a user interface that allows for text expansion during translation. To avoid translation problems, consider the topics in the following sections:

- Text Expansion and Contraction
- Message Text
- Layout

You cannot rely on translation to make the user interface audience specific. You might have to design an audience-specific interface.

5.4.1 Text Expansion and Contraction

One of the biggest considerations during the design phase is the translation of text into another language. Some of the translation issues you must consider are discussed in the following sections:

- Word Length
- Typefaces and Point Sizes
- Word Order

Word Length

Failure to allow for changes in size and length of words in an interface is a common error that can be costly during the localization process. If you design an interface that uses every available space or widgets that are exactly the right size for their labels, you are not allowing for the growth of some words during the translation process. For example, when you translate an application from English to German, you should plan on significant expansion of text in specific places, such as button labels.

Table 5–3 illustrates the difference in text length when you translate the text from one language to another.

Table 5–3: The Difference In Text Length After Translation

Language	Translation Of “Message Pop-up”
Danish	pop-up-meddelelse
Danish	berichtvenster
English	message popup
Finnish	sanomakohovalikko
French	incrustation de message
German	Nachrichtenüberlagerungsfenster
Italian	messaggio sovrapposto
Norwegian	meldingsvindu
Portuguese	janela de sobreposiçao de mensagem
Spanish	mensaje emergente
Swedish	popupmeddelande

Typefaces and Point Sizes

When you translate to a different language, the typeface and point size might change. For example, translating from English to Chinese requires a different typeface (to represent the ideograms) and a larger point size (so that the ideograms are legible). For this reason, you must ensure that fields and labels to be translated reside in widgets large enough to accommodate the change in typeface and point size.

NOTE: In general, double-byte Asian languages require larger point sizes than do other languages.

Word Order

Different languages use different word orders. For example, in English, adjectives precede the nouns they modify. In French, some adjectives might have a different meaning depending on the context and whether the adjective precedes or follows the noun it modifies. If you write the phrase “world population conference,” the translator might not be able to determine whether the phrase means a conference on the population of the world or a conference on population in general with attendees from all over the world. If you design an application in French and localize to an English-speaking country, you must review and possibly adjust any labels, titles, or messages that contain adjective and noun combinations.

You should account for such adjustments when you design your interface and write your messages. To save time in translation, use variables to define labels, titles, or messages, and plan to make adjustments for translation in only one place. In some cases, you can use the SUBSTITUTE function to create phrases that you can reorganize in the translation process by using different substitution arguments for each portion of a phrase. See the *Progress Language Reference* for more information on the SUBSTITUTE function.

5.4.2 Message Text

Translating error messages and status bar messages can present unique problems during localization. It is extremely important that you do not build error messages from separate pieces of text. When you build messages, the context of the words is lost and, in some instances, gender and case disagree, making it very difficult for the translator to translate your messages.

Messages sometimes have multiple translations. Developers should create messages that are clear, concise, and contain useful descriptions of the problem that is causing the error. Also, do not use embedded SKIP options in messages. When the message is translated, the alignment might look unusual.

5.4.3 Layout

Design menu bars, status bars, toolbars, title bars, and dialog boxes to allow for text size to increase. Dialog boxes that are full of information will not fit on the screen when translated into some languages. Use multi-line text boxes to display either messages or multiple-word descriptions that might require additional space when translated.

Limit the number of menus on your menu bar so that it does not wrap to a second line when translated.

NOTE: Translation might affect the placement of objects as well as text on the user interface.

6

Using Databases

Using databases in applications you intend to internationalize and localize is not difficult. It only requires following a few rules and knowing a few techniques. This chapter discusses the rules and techniques.

Specifically, this chapter contains the following sections:

- Progress Database Name Restrictions
- Empty Databases
- Scanning Databases For Character Conflicts
- Converting a Database and Its Data To a Different Code Page
- Loading Table Dump Files
- Specifying -cpinternal and -cpstream With Database Utilities

6.1 Progress Database Name Restrictions

Database data can contain non-ASCII characters.

The rules are different, however, for database names, which include the logical and physical name of the database as well as names of tables, fields, indexes, and sequences. Database names can contain any combination of English letters and numbers, but they must begin with a letter from A–Z or a–z. They cannot include any of the following:

- Progress 4GL reserved words
- Letters with diacritical marks
- The following characters:

```
\ " ' * ; | ? [ ] ( ) ! { } < >
```

For more information on Progress database name restrictions, see the *Progress Database Administration Guide and Reference*. For more information on Progress 4GL reserved words, see the *Progress Language Reference*.

6.2 Empty Databases

As you develop an application to be deployed across multiple locales, you might need an empty database in a particular code page. Progress supplies a collection of these. You can also create your own.

6.2.1 Empty Databases Supplied By Progress

Progress supplies a collection of empty databases in a variety of code pages. These empty databases reside in the `DLC/prolang` directory in subdirectories by locale. For example, the `DLC/prolang/jpn` directory contains empty Japanese databases and the `DLC/prolang/tur` directory contains empty Turkish databases.

For more information on the contents of the `DLC/prolang` directory, see the `DLC/prolang/readme` file.

6.2.2 Creating an Empty Database In a Particular Code Page

Here is another technique for producing an empty database in a particular code. The technique is included here for completeness. To create an empty database with a code page, follow these steps:

1 ♦ Create an empty database, using the Progress Data Dictionary utility.

2 ♦ Within the Progress distribution, select the `DLC/prolang` directory for the target locale.

For example, the `DLC/prolang/cze` directory contains files used for Czech databases, while the `DLC/prolang/utf` directory contains files used for Unicode UTF–8 databases.

3 ♦ Within the `DLC/prolang` directory you selected, select the collation data definition (`.df`) file whose name corresponds to the target locale.

These `.df` files define database collations (as opposed to `.df` files that define database schemas). For example, the `DLC/prolang/cze` directory contains the following collation data definition files:

- `cze1250.df`, associated with the CZECH collation of code page 1250
- `cze852.df`, associated with the CZECH collation of code page 852
- `cze8859.df`, associated with the CZECH collation of code page ISO8859–1

4 ♦ Confirm the collation name of the `.df` file you selected in Step 3 by viewing its contents using a text editor. Search for the COLLATION-NAME keyword. The character string that follows is the name of the `.df` file's collation.

5 ♦ Load the `.df` file whose collation name you confirmed in Step 4 into the empty database you created in Step 1 using the following steps:

a) Using the Data Administration tool, choose Admin→ Load Data and Definitions (`.df` file). The Load Data Definitions dialog box appears.

b) Enter the name of the data definition file into the Input File field, then choose OK.

NOTE: The Data Administration tool calls `.df` files collation tables. The default is `_tran.df`.

6 ♦ Rebuild the database's indexes by running the PROUTIL utility with the IDXBUILD qualifier. The syntax is:

SYNTAX

```
proutil db-name -C idxbuild all
  -cpinternal internal-code-page
  -cpstream stream-code-page
```

For the complete syntax of the PROUTIL utility, see the *Progress Database Administration Guide and Reference*.

The empty database is now associated with the desired code page.

6.3 Scanning Databases For Character Conflicts

You might want to convert a database and its data to a different code page. But before you do this, you must determine if the code-page conversion you are planning involves a character conflict.

Consider the conversion from IBM850 to ISO8859–1. Every character that appears in IBM850 appears in ISO8859–1. That is, this code-page conversion does not involve a character conflict.

But consider the conversion from ISO8859–1 to ISO8859–15. ISO8859–1 contains characters that ISO8859–15 does not. If you are considering this conversion, you must determine if an ISO8859–1 database contains any character not in ISO8859–15. If it does, there is a character conflict.

6.3.1 New Characters

Code pages 1252 and ISO8859–15 contain characters not in ISO8859–15. Table 6–1 lists these:

Table 6–1: New Characters In 1252 and ISO8859–15

Character	Position in 1252	Position in ISO8859–15
Euro symbol	128	164
Š (S with caron)	138	166
š (s with caron)	154	168
Ž (Z with caron)	142	180
ž (z with caron)	158	184
Œ (OE ligature)	140	188
œ (oe ligature)	156	189
Ÿ (Y with dieresis)	159	190

Table 6–2 lists the characters to check for before converting a database to 1252. Each row of the table lists values for a particular source code page. For example, before converting from IBM037, check for the characters in the second row.

Table 6–2: Characters To Check For Before Converting To MS1252

Code Page	Euro	Š	Œ	Ž	š	œ	ž	Ÿ
1252	128	138	140	142	154	156	158	159
IBM037	33	43	45	47	59	61	63	255
IBM273	32	42	44	46	58	60	62	63
IBM277	33	43	45	47	59	61	63	255
IBM278	33	43	45	47	59	61	63	255
IBM284	33	43	45	47	59	61	63	255
IBM297	33	43	45	47	59	61	63	255
IBM437	158	183	185	187	199	201	203	204
IBM500	33	43	45	47	59	61	63	255
IBM850	213	192	194	196	219	223	254	159
IBM858	213	192	194	196	219	223	254	159
IBM861	158	184	186	188	200	202	204	205
ISO8859–1	128	138	140	142	154	156	158	159

Table 6–3, which has the same format as Table 6–3, lists the characters to check for before converting a database to ISO8859–15. Each row lists values for a particular source code page. For example, before converting from IBM037, check for the characters in the second row.

Table 6–3: Characters To Check For Before Converting To ISO8859–15

Code Page	Euro	Š	Œ	Ž	š	œ	ž	Ÿ
ISO8859–15	164	166	168	180	184	188	189	190
IBM037	159	106	189	190	157	183	184	185
IBM273	159	204	189	190	157	183	184	185
IBM277	90	122	189	190	157	183	184	185
IBM278	90	204	189	190	157	183	184	185
IBM284	159	73	161	190	157	183	184	185
IBM297	159	221	161	190	157	183	184	185
IBM437	206	207	209	215	217	172	171	219
IBM500	159	106	189	190	157	183	184	185
IBM850	213	221	249	239	247	172	171	243
IBM858	213	221	249	239	247	172	171	243
IBM861	16	178	209	215	216	172	171	219
ISO8859–1	164	166	168	180	184	188	189	190
ROMAN8	186	169	171	168	236	247	248	245

6.3.2 Using PROUTIL To Perform the Scan

When you are considering a particular code-page conversion, you want to check an existing database for characters that do not appear in the target code page. This is precisely what is done by the PROUTIL utility with the CONVCHAR CHARSCAN qualifier. It searches for every occurrence of specified characters in every CHARACTER field in the database, using the code page associated with the database. It reports the total number of matched fields, and for each matched field, reports the table name, field name, and record id.

The syntax is:

SYNTAX

```
proutil db-name -C convchar charscan target-code-page "character-list"
  -cpinternal internal-code-page
  -cpstream stream-code-page
```

db-name

The database to be converted.

target-codepage

The code page you are considering converting to.

character-list

A list of characters. The character list consists of code-page positions separated by commas. The string can include decimal or hexadecimal values for up to ten characters.

NOTE: For the complete syntax of the PROUTIL utility, see the *Progress Database Administration Guide and Reference*.

Here are some examples of scanning databases for character conflicts:

- You have an ISO8859–1 database called mydb that you are thinking of converting to 1252. To scan this database for the presence of characters not in 1252, run the following command:

```
proutil mydb -C convchar charscan 1252 "128,138,140,142,154,156,158,159"
```

- You have an IBM437 database called mydb that you are thinking of converting to ISO8859–15. To scan this database for the presence of characters not in ISO8859–15, run the following command:

```
proutil mydb -C convchar charscan ISO8859-15
"206,207,209,215,217,172,171,219"
```

6.4 Converting a Database and Its Data To a Different Code Page

You can convert a database and its data from one code page (the *source* code page) to another code page (the *target* code page) if one or both of the following conditions are true:

- Every character that appears in the source code page appears in the target code page.
- Every character that appears in the database appears in the target code page.

You might want to convert a database and its data to a particular code page if the current code page does not contain characters that the application needs and that another code page contains. For example, code pages 1252 and ISO8859–15 contain characters that ISO8859–1 does not, as you saw in Table 6-1.

Another reason is to convert a database and its data to a particular code page is if the database's data comes exclusively from a device with that code page. For example, if an airline reservations database gets its data exclusively from a terminal that uses the ISO8859–1 code page, you might want to convert the database and its existing data to ISO8859–1.

To convert a database and its data to another code page, run the PROUTIL utility with the CONVCHAR CONVERT qualifier. The syntax is:

SYNTAX

```
proutil dbname -C convchar convert new-codepage-name
```

dbname

The name of the database.

new-codepage-name

The name of the target code page.

For more information on code-page conversion, see Chapter 2, "Understanding Code Pages." For the complete syntax of the PROUTIL utility, see the *Progress Database Administration Guide and Reference*.

6.5 Loading Table Dump Files

When a table dump file loads, it is essential that Progress interprets the table dump file's data correctly—that is, according to the code page the data was encoded in. If Progress interprets the data using a code page other than the one the data was encoded in, the database is populated with garbled data. This issue is especially critical in environments with multiple code pages.

6.5.1 Code-page Trailers

Some table dump files contain code-page trailers, which tell Progress the code page of the table dump file's data. When you load such a table dump file, Progress reads the code-page trailer, determines the code page of the file's data, and interprets the data correctly. In the current version of Progress, when you dump a database using the Data Dictionary utility, the resulting table dump file contains a code-page trailer. Figure 6–1 shows a typical code-page trailer with the code page name "IBM850."

```
      .
      .
      .
PSC
filename=customer
records=00000033
ldbname=xl
timestamp=1994/03/30-10:20:03
numformat=.
dateformat=mdy-1900
cpstream=ibm850
      .
      .
      .
0000006570
```

Figure 6–1: A Typical Code-page Trailer

Other table dump files do not contain code-page trailers. When you load such a table dump file, you must tell Progress the code page of the file's data by using the -cpstream startup parameter. To determine the code page of a table dump file that lacks a code-page trailer, see Chapter 2, "Understanding Code Pages."

6.5.2 Techniques For Loading Table Dump Files

There are several techniques for loading a table dump file. The techniques include using a 4GL program, using the Data Dictionary utility, and using the PROUTIL utility with the BULKLOAD qualifier.

Using a 4GL Program

You can load a table dump file using a simple 4GL program such as the following, which loads the table dump file `x.d`:

```
INPUT FROM x.d.
REPEAT:
  CREATE x.
  IMPORT x.
END.
```

NOTE: A 4GL program such as the preceding cannot process table dump files with code page trailers.

If you load the table dump file using this program, which uses the IMPORT statement, you probably dumped the database using a companion 4GL program that uses the EXPORT statement. The resulting table dump file does not contain a code page trailer.

Using the Data Dictionary Utility

You can load a table dump file using the Data Dictionary utility, which has graphical and character versions. For more information on the graphical version, see its online help. For more information on the character version, see the *Progress Basic Development Tools* manual.

Using the PROUTIL Utility With the BULKLOAD Option

You can load a table dump file using the PROUTIL utility with the BULKLOAD qualifier. The syntax is:

SYNTAX

```
proutil db-name [ -yy n ] -C BULKLOAD fd-file [ -Bn ]
```

For the complete syntax of the PROUTIL utility, see the *Progress Database Administration Guide and Reference*.

6.6 Specifying -cpinternal and -cpstream With Database Utilities

When you install Progress and specify default values for -cpinternal and -cpstream, the installation program writes these values to `DLC/startup.pf`, the main parameter file. You you subsequently run Progress or a database utility without specifying a value for -cpinternal or -cpstream, Progress uses the value in the main parameter file.

7

Using SQL-92

Progress provides several programming languages. One of these is SQL-92, the focus of this chapter.

This chapter contains the following sections:

- Starting Database Servers
- Using SQL-92 Database Servers
- Code-page Conversion In SQL-92 Applications
- Using SQL-92 Clients
- Using SQL-92 Utilities
- Using the SQL-92 Language

NOTE: For a complete description of the Progress SQL-92 language, see the *Progress SQL-92 Guide and Reference* and the *Progress Embedded SQL-92 Guide and Reference*.

7.1 Starting Database Servers

You can start a database server in two ways:

- By invoking PROSERVE from a command line or from a Progress client
- By using Progress Explorer, a graphical application

When a Progress client invokes PROSERVE, PROSERVE starts a database server on behalf of the client. For a 4GL client, PROSERVE starts a 4GL database server. For an SQL-92 client, PROSERVE starts an SQL-92 database server.

If a client invokes PROSERVE with startup parameters, such as -cpinternal, -cpstream, and -cpcoll, PROSERVE passes these values to the 4GL or SQL-92 database server. The 4GL database server uses these values. The SQL-92 database server, however, ignores some of these values. Specifically, the SQL-92 database server ignores values for -cpinternal and -cpcoll passed from PROSERVE, using instead the internal code page and collation table of the database. However, the SQL-92 database server does use the values for -cpstream and for the other code-page-related startup parameters (other than -cpinternal and -cpcoll) passed from PROSERVE.

When you start a database server using Progress Explorer, you can specify a code page and a collation by using the Explorer's administrative features.

Table 7–1 lists localizable startup parameters and settings related to SQL-92 and 4GL database servers.

Table 7–1: Localizable Startup Parameters and Settings

Command Line Startup Parameters	Command Line Startup Parameter Defaults	Progress Explorer Settings	Progress Explorer Setting Defaults
-cpinternal	ISO8859-1	Code Page (internal)	ISO8859-1
-cpstream	IBM850	Not Applicable	IBM850
-cplog	IBM850	Log Character Set	ISO8859-1
-cpcase	BASIC	Case Table	BASIC
-cpcoll	BASIC	Collation Table	BASIC
-convmap	`DLC/convmap.cp`	ConversionMap	`DLC/convmap.cp`

Progress Explorer also lets you set the log file's font. This allows you to choose a font that displays international characters correctly. To set the log file's font, follow these steps:

1 ♦ From Progress Explorer, view a log file.

2 ♦ From the Action menu or popup, select "Font...". Alternatively, on the toolbar, click on the Font icon.

The standard Windows font selection dialog box appears.

3 ♦ In the Font Selection dialog box, choose a font, then click OK.

Progress Explorer displays the log file using the font you selected.

7.2 Using SQL-92 Database Servers

When you use SQL-92 database servers, the following notes apply:

- The SQL-92 database server is single threaded. This means that each client connection requires one server process.
- The SQL-92 server uses the current Progress `convmap.cp` file. This is the one that the PROCONV environment variable, if set, points to. Else, this is the one in the `DLC` directory.
- The SQL-92 server uses the code page of the connected database as its internal code page. Similarly, the SQL-92 server uses the collation table of the connected database as its internal collation table. This means that all queries executed on the server (as opposed to on the client) depend on the internal code page and the internal collation table of the database.
- The SQL-92 database server uses the same PROMSGS file as PROSERVE. This file is the one specified by the PROMSGS environment variable, if set. Else, this file is the `promsgs` file in the `DLC` directory.

7.3 Code-page Conversion In SQL-92 Applications

In SQL-92 applications, when Progress performs code-page conversions, these generally occur on the server. In 4GL applications, by contrast, these generally occur on the client.

Figure 7–1 shows a typical SQL-92 application and the code-page conversions performed by Progress.

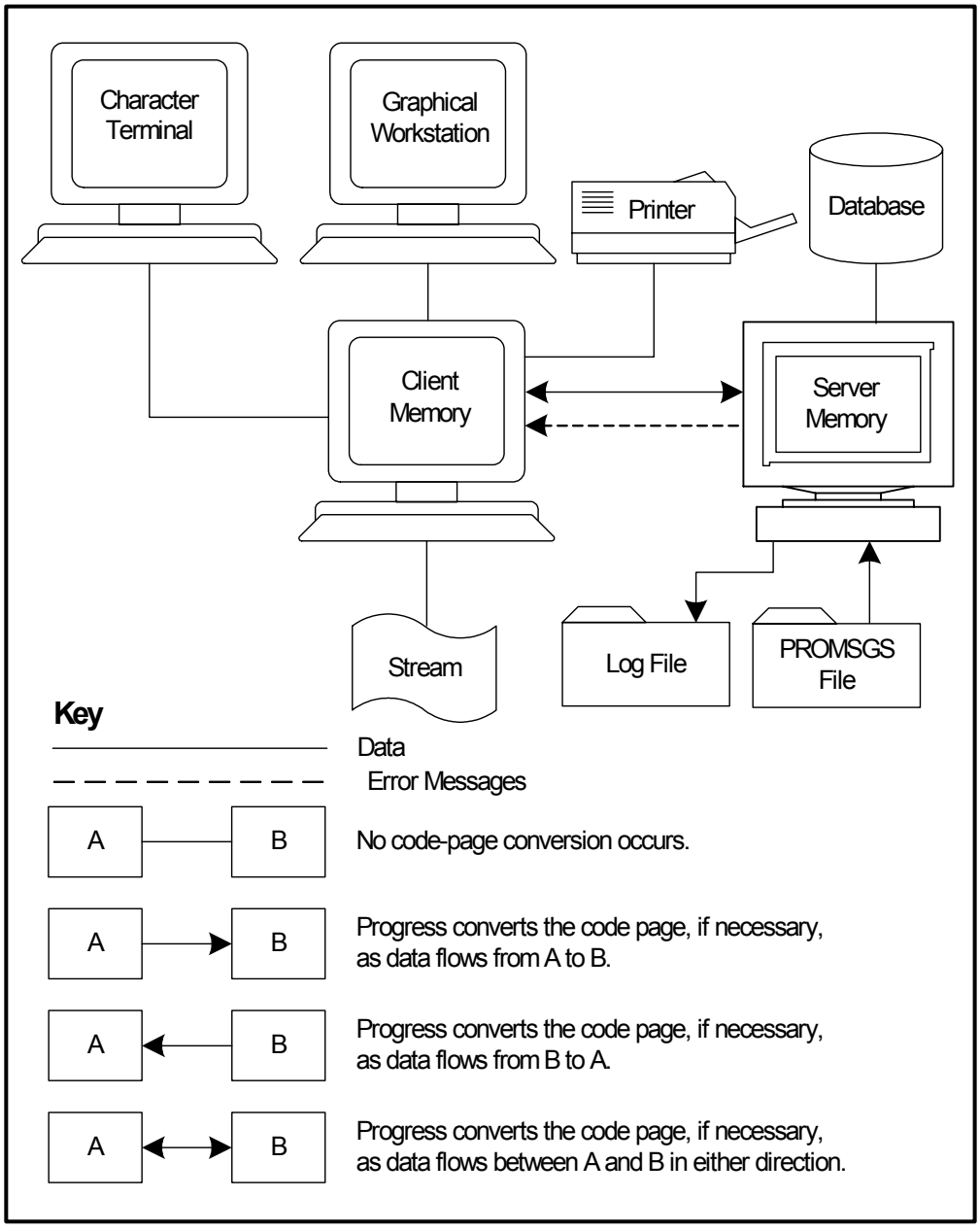

Figure 7–1: Code-page Conversion In an SQL-92 Application

Compare Figure 7–1 with Figure 7–2, which shows a typical 4GL application and the code-page conversions performed by Progress.

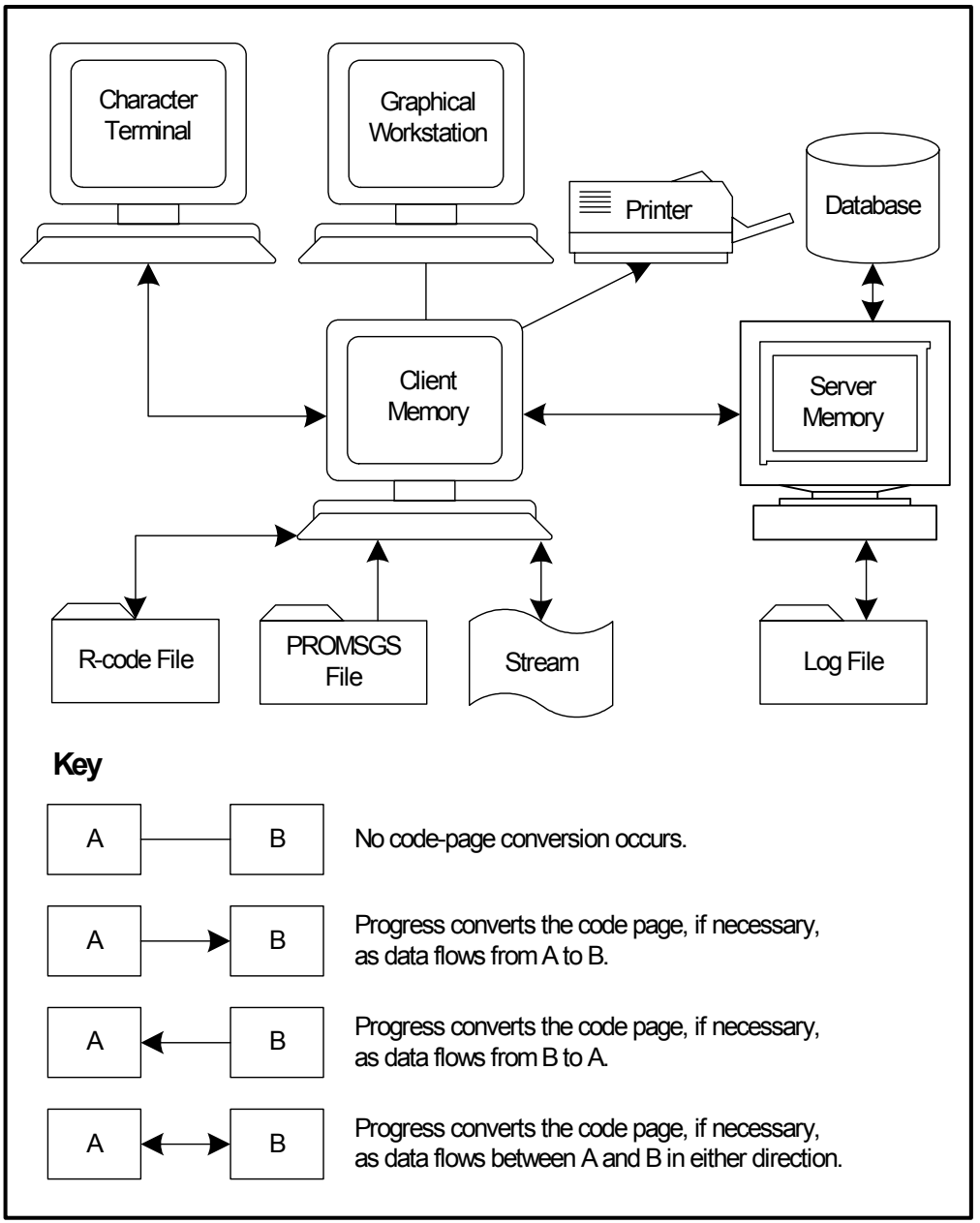

Figure 7–2: Code-page Conversion In a 4GL Application

7.4 Using SQL-92 Clients

SQL-92 databases can have the following types of client:

- Embedded SQL-92 (ESQL-92)
- Java/Java Database Connectivity (Java/JDBC)
- Open Database Connectivity (ODBC)

7.4.1 ESQL-92 Clients

ESQL-92 clients, in source code form, consist of C language statements with ESQL-92 statements interleaved. To prepare an ESQL-92 program for execution, preprocess it using the Progress ESQL-92 preprocessor, compile it using a C compiler, and link it, making sure the link includes the Progress ESQL-92 libraries.

Examples of ESQL-92 clients include the SQLDUMP, SQLLOAD, and SQLSCHEMA utilities. For more information on these utilities, see the "Using SQL-92 Utilities" section. For more information on building ESQL-92 clients, see the *Progress Embedded SQL-92 Guide and Reference*.

7.4.2 Java/JDBC Clients

Java/JDBC clients, in source code form, consist of statements in the Java language. Java/JDBC clients communicate with SQL-92 databases by following the JDBC interface guidelines. To prepare a Java/JDBC program for execution, compile it using a Java compiler.

An example of a Java/JDBC client is the Progress SQL Explorer. For more information on building Java/JDBC clients, see the *Progress JDBC Driver Guide*.

7.4.3 ODBC Clients

ODBC clients access Progress SQL-92 databases by following the ODBC interface guidelines. At run time, ODBC clients access the ODBC driver, which consists of a Windows DLL (dynamic link library).

For more information on building ODBC clients, see the *Progress ODBC Driver Guide*.

7.4.4 Characteristics Of SQL-92 Clients

Progress SQL-92 clients have:

- An internal code page
- A code page for displaying PROMSGS sent by the server to the client

Internal Code Page

For Java/JDBC clients, the internal code page is Unicode, which is converted to UTF–8, an encoding of Unicode, to communicate with the JDBC driver. For more information on Unicode and UTF–8, see Chapter 9, "Using Unicode."

For ESQL-92 and ODBC clients, the internal code page is the value of the client's SQL_CLIENT_CHARSET environment variable, if set. Else, the internal code page is that of the client's locale. For more information on the SQL_CLIENT_CHARSET environment variable, see the *Progress Installation and Configuration Guide Version 9 for UNIX* or the *Progress Installation and Configuration Guide Version 9 for Windows*.

Code Page For Displaying PROMSGS Sent By the Server To the Client

When PROMSGS messages sent by the database server arrive at the client, the client accesses them using a particular code page. This code page is the one named by the client's SQL_CLIENT_CHARSET_PROMSGS environment variable, if set. Else, this code page is the one named by the client's SQL_CLIENT_CHARSET environment variable, if set. Else, this code page is that of the client's locale.

7.5 Using SQL-92 Utilities

Progress provides several utilities for working with SQL-92 databases. These include:

- SQLDUMP
- SQLLOAD
- SQLSCHEMA

7.5.1 SQLDUMP

SQLDUMP, an ESQL/C application, lets you dump SQL-92 databases.

The syntax is:

Operating System	Syntax
UNIX Windows	`sqldump -u` *user_name* **[** `-a` *password* **]** **[** `-C` *code-page-name* **]** `-t` **[** *owner_name.***]***table_name1* **[[**,*owner_name.***]***table_name2*, **...** **]** *database_name*

NOTE: For the complete syntax of SQLDUMP, see the *Progress Database Administration Guide and Reference*.

-C `code-page-name`

A case-insensitive character string that specifies the name of the dump file's code page. If the -C parameter specifies a code page name that is not valid, Progress reports a run time error. If the -C parameter does not appear at all, the code page name defaults to that of the client's internal code page, which is:

- The value of the client's SQL_CLIENT_CHARSET environment variable, if set
- Otherwise, the name of the code page of the client's locale.

You might use the -C parameter. for example, to have a Windows client using the MS1250 code page produce a dump file using the ISO8859–2 code page (say, to read later on a UNIX machine). Although you could accomplish this by setting the client's SQL_CLIENT_CHARSET environment variable, using the -C parameter does not affect other clients.

NOTE: By default, SQLDUMP displays PROMSGS messages from the server using the code page specified by `code-page-name`—unless you specify a different code page, which you can do by setting the client's SQL_CLIENT_CHARSET_PROMSGS environment variable.

7.5.2 SQLLOAD

SQLLOAD, an ESQL/C application, lets you load SQL-92 databases.

The syntax for SQLLOAD is:

Operating System	Syntax
UNIX Windows	`sqlload -u` *user_name* **[** `-a` *password* **]** `-t` **[** *owner_name.***]***table_name1* **[[**`,`*owner_name.***]***table_name2*`, . . .` **]** **[** `-l` *log_file_name* **]** **[** `-b` *badfile_name* **]** **[** `-e` *max_errors* **]** **[** `-s` *skipcount* **]** **[** `-m` *maxrows* **]** **[** `-F comma` **|** `quote` **]** *database_name*

NOTE: For the complete syntax of SQLLOAD, see the *Progress Database Administration Guide and Reference*.

-C code-page-name

A case-insensitive character string that specifies the name of the code page of the client. If the -C parameter specifies a code page name that is invalid, Progress reports a run time error. If the -C parameter does not appear at all, the code page name defaults to that client's internal code page, which is:

- The value of the client's SQL_CLIENT_CHARSET environment variable, if set
- Otherwise, the name of the code page of the client's locale.

You might use the -C parameter, for example, to load a dump file whose code page is ISO8859-2 using a Windows client whose code page is MS1250. Although you could accomplish this by setting the client's SQL_CLIENT_CHARSET environment variable, using the -C parameter does not affect other clients.

At run time, SQLLOAD reports an error if the code page of the dump file does not match the internal code page of the client.

NOTE: By default, SQLLOAD displays PROMSGS messages using the code page corresponding to *code-page-name*—unless you specify a different code page, which you can do by setting the client's SQL_CLIENT_CHARSET_PROMSGS environment variable.

7.5.3 SQLSCHEMA

SQLSCHEMA is a command-line utility that writes selected components of an SQL-92 database schema to an output file using the UTF-8 code page. You might then use the Progress SQL Explorer to load the schema components into an SQL-92 database

The syntax for SQLSCHEMA is:

Operating System	Syntax
UNIX Windows	sqlschema -u *user_name* [-a *password*] [-t [*owner_name.*]*table_name1* [,*owner_name.*]*table_ name2*,...] [-p [*owner_name.*] *procedure_name*, ...] [-T [*owner_name.*]*trigger_name*, ...] [-g [*owner_name.*]*table_name*, ...] [-s [owner_name.]*table_name*, ...] [-o *output_file_name*] *database_name*

For the complete syntax of SQLSCHEMA, see the *Progress Database Administration Guide and Reference*.

7.6 Using the SQL-92 Language

When you use the SQL-92 language in applications you intend to localize to multiple locales, the following points might help you:

- The unit of length when working with character strings
- The maximum number of bytes required by items of type CHAR and VARCHAR
- SQL-92 elements that support internationalization and localization
- Format specifiers allowed with the TO_CHAR() and TO_DATE() functions

For a complete list of the international elements in Progress SQL-92, see Appendix A, "Progress Resources." For a complete description of the Progress SQL-92 language, see the *Progress SQL-92 Guide and Reference* and the *Progress Embedded SQL-92 Guide and Reference*.

7.6.1 The Unit Of Length When Working With Character Strings

The string operators in Progress SQL-92 consider the unit of length to be the character count, not a byte count or a column count.

Consider the syntax of the Progress SQL-92 LEFT function:

SYNTAX

```
LEFT ( character-string-expression , length )
```

LEFT evaluates *character-string-expression* and returns the leftmost *length* characters, whether single byte, double byte, or triple byte.

In the following expression, *character-string-expression* consists of a four characters: the first single byte. the second double byte, and the third and fourth single byte:

```
LEFT("A中BC",3)
```

The preceding expression returns the following string, which consists of three characters, the second of which is double byte, for a total of four bytes:

```
A中B
```

7.6.2 The Maximum Number Of Bytes Required By Fields Of Type CHAR and VARCHAR

In Progress SQL-92, when you create a field of type CHAR or VARCHAR, the maximum number of bytes required depends on whether the code page is single-byte, double-byte, or triple-byte. For example, a CHAR(4) field requires a maximum of 4 bytes if the code page is single byte, a maximum of 8 bytes if the code page is double-byte, and a maximum of 12 bytes if the code page is triple-byte.

NOTE: If you define a field as CHAR(10000) and the code page is UTF–8, which is a triple byte, the field requires a maximum of 30,000 bytes of storage.

7.6.3 SQL-92 Elements That Support Internationalization and Localization

The Progress SQL-92 elements that support internationalization and localization appear in Figure 7–2.

Table 7–2: SQL-92 Language Elements That Support Internationalization and Localization *(1 of 11)*

Syntax	Description	Comment
{ CHARACTER | CHAR } [(*length*)]	The CHARACTER data type represents a null-terminated character field of length *length*.	*length* represents the number of characters.
{ CHARACTER VARYING | CHAR VARYING | VARCHAR } [(*length*)]	The VARCHAR data type represents a variable-length character field whose maximum length is *length*.	*length* represents the maximum number of characters.
= | <> | != | ^= | < | <= | > | >=	Relational operators specify how SQL compares expressions in basic and quantified predicates.	To sort and compare CHARACTER data, the SQL server uses the numeric values in the collation table, not the numeric values in the code page.

Table 7–2: SQL-92 Language Elements That Support Internationalization and Localization *(2 of 11)*

Syntax	Description	Comment
column_name [NOT] LIKE *string_constant* [ESCAPE *escape_character*]	The LIKE predicate searches for strings that have a certain pattern. The pattern is specified after the LIKE keyword in a string constant. The pattern can be specified by a string in which the underscore (_) and percent sign (%) characters have special semantics.	The LIKE predicate is multi-byte enabled. The *string_constant* and the *escape_character* may contain multi-byte characters, and *escape_character* can be a multi-byte character. A percent sign (%) or an underscore (_) in *string_constant* can represent a multi-byte character. However, the percent sign or underscore itself must be the single-byte ASCII encoding. The comparison is case-insensitive.
'*char-string*'	A character-string literal is a string of characters enclosed in single quotation marks (' '). To include a single quotation mark in a character-string literal, precede it with an additional single quotation mark.	A character string literal may contain multi-byte characters in the character set used by the SQL client. Only single-byte ASCII-encoded quote marks are valid in the syntax.

Table 7–2: SQL-92 Language Elements That Support Internationalization and Localization *(3 of 11)*

Syntax	Description	Comment

{ d '*yyyy-mm-dd*' }
  | *mm-dd-yyyy*
  | *mm/dd/yyyy*
  | *mm-dd-yy*
  | *mm/dd/yy*
  | *yyyy-mm-dd*
  | *yyyy/mm/dd*
  | *dd-mon-yyyy*
  | *dd/mon/yyyy*
  | *dd-mon-yy*
  | *dd/mon/yy*
{ t '*hh:mi:ss*' }
  | *hh:mi:ss*[:*mls*]
	SQL supports special formats for date and time literals. Basic predicates and the VALUES clause of INSERT statements can specify date literals directly for comparison and insertion into tables. In other cases, you need to convert date literals to the appropriate date-time data type with the CAST, CONVERT, or TO_DATE scalar functions	All text (names of days, months, ordinal number endings) in all types date-format literals must be in the English Language. The default date format is American. You can explicitly request another date format by using a format string.Time literals are in the English Language only.

ASCII (*char_expression*)	The scalar function ASCII returns the ASCII value of the first character of the given character expression.	The ASCII function depends on the code page and supports multi-byte characters. The function returns the character encoding integer value of the first character of *char_expression* in the current code page. Whether *char_expression* represents a literal string or a database field, the result depends on the code page of the database.

Table 7–2: SQL-92 Language Elements That Support Internationalization and Localization *(4 of 11)*

Syntax	Description	Comment
CHAR (*integer_expression*)	The scalar function CHAR returns a character string with the first character having an ASCII value equal to the argument expression. CHAR is identical to CHR but provides ODBC-compatible syntax.	The CHAR function depends on the code page and supports single-byte and multi-byte characters. If *integer_expression* evaluates to an integer value that represents a character in the database code page, CHAR returns that character. Else, CHAR returns a NULL value.
CHR (*integer_expression*)	The scalar function CHR returns a character string with the first character having an ASCII value equal to the argument expression.	The CHR function depends on the code page and supports single-byte and multi-byte characters. If *integer_expression* evaluates to an integer value that represents a character in the database code page, CHR returns that character. Else, CHR returns a NULL value.
CONCAT (*char_expression* , *char_expression*)	The scalar function CONCAT returns a concatenated character string formed by concatenating argument one with argument two.	The two *char_expression* expressions and the result of the CONCAT function can contain multi-byte characters.

Table 7–2: SQL-92 Language Elements That Support Internationalization and Localization *(5 of 11)*

Syntax	Description	Comment
CONVERT (*'data_type* [(*length*)] *' , expression*)	The Progress Extension scalar function CONVERT converts an expression to another data type. The first argument is the target data type. The second argument is the expression to be converted to that type.	When *data_type* is CHARACTER(*length*) or VARCHAR(*length*), the *length* specification represents the number of characters. The converted result can contain multi-byte characters.
GREATEST (*expression* , *expression* ...)	The scalar function GREATEST returns the greatest value among the values of the given expressions.	When the data type of an *expression* is either CHARACTER(*length*) or VARCHAR(*length*), the expression can contain multi-byte characters. The sort weight for each character is determined by the collation table in the database.
INITCAP (*char_expression*)	The scalar function INITCAP returns the result of the argument character expression after converting the first character to uppercase and the subsequent characters to lowercase.	A *char_expression* and the result can contain multi-byte characters. To convert the first character to uppercase and the subsequent characters to lowercase, Progress uses a case table in the `convmap.cp` file. The default case table is BASIC.

Table 7–2: SQL-92 Language Elements That Support Internationalization and Localization *(6 of 11)*

Syntax	Description	Comment
INSERT (*string_exp1* , *start_pos* , *length* , *string_exp2*)	The scalar function INSERT returns a character string where *length* number of characters have been deleted from *string_exp1* beginning at *start_pos*, and *string_exp2* has been inserted into *string_exp1*, beginning at *start_pos*.	*string_exp1*, *string_exp2*, and the result might contain multi-byte characters, depending on the code page of the SQL server. The *length* argument specifies a character count.
INSTR (*char_expression1* , *char_expression2* [, *start_pos* [, *occurrence*]])	The INSTR (in string) scalar function searches character string *char_expression1* for the character string *char_expression2*. The search begins at *start_pos* of *char_expression1*. If *occurrence* is specified, then INSTR searches for the *n*th occurrence, where *n* is the value of the fourth argument.	A *char_expression* and the result can contain multi-byte characters.
LCASE (*char_expression*)	The scalar function LCASE returns the result of the argument character expression after converting all the characters to lowercase. LCASE is the same as LOWER but provides ODBC-compatible syntax.	A *char_expression* and the result may contain multi-byte characters. The conversion to lowercase conversion depends on the case table in the `convmap` file. The default case table is BASIC.

Table 7–2: SQL-92 Language Elements That Support Internationalization and Localization *(7 of 11)*

Syntax	Description	Comment
LEAST (*expression* , *expression* , ...)	The scalar function LEAST returns the lowest value among the values of the given expressions.	When the data type of an *expression* is either CHARACTER(*length*) or VARCHAR(*length*), the *expression* may contain multi-byte characters. The sort weight for each character depends on the collation table in the database.
LEFT (*string_exp* , *count*)	The scalar function LEFT returns the leftmost count of characters of *string_exp*.	The *string_exp* and the result can contain multi-byte characters. The function returns a character count.
LENGTH (*char_expression*)	The scalar function LENGTH returns the string length of the value of the given character expression.	*char_expression* may contain multi-byte characters. The function returns a character count.
LOCATE (*char_expr1* , *char_expr2* , [*start_pos*])	The scalar function LOCATE returns the location of the first occurrence of *char_expr1* in *char_expr2*. If the function includes the optional integer argument *start_pos*, LOCATE begins searching *char_expr2* at that position. If the function omits the *start_pos* argument, LOCATE begins its search at the beginning of *char_expr2*.	*char_expr1* and *char_expr2* can contain multi-byte characters. The *start_pos* argument specifies a character position, not a byte position. Character comparisons use the collation table in the database.

Table 7–2: SQL-92 Language Elements That Support Internationalization and Localization *(8 of 11)*

Syntax	Description	Comment
LPAD (*char_expression* , *length* [, *pad_expression*])	The scalar function LPAD pads the character string corresponding to the first argument on the left with the character string corresponding to the third argument. After the padding, the length of the result is *length*.	The *char_expression* and *pad_expression* can contain multi-byte characters. The *length* specifies a number of characters.
LTRIM (*char_expression* [, *char_set*])	The scalar function LTRIM removes all the leading characters in *char_expression* that are present in *char_set* and returns the resulting string. The first character in the result is guaranteed not to be in *char_set*. If you do not specify the *char_set* argument, leading blanks are removed.	The *char_expression*, the character set specified by *char_set*, and the result can contain multi-byte characters. Character comparisons are case-sensitive and depend on the collation table in the database.
PREFIX (*char_expression* , *start_pos* , *char_expression*)	The scalar function PREFIX returns the substring of a character string, starting from the position specified by *start_pos* and ending before the specified character.	Each *char_expression* and the result can contain multi-byte characters. The *start_pos* argument specifies a character position, not a byte position. Character comparisons are case-sensitive and depend on sort weights in the collation table in the database.
REPEAT (*string_exp* , *count*)	The scalar function REPEAT returns a character string composed of *string_exp* repeated *count* times.	*string_exp* and the result can contain multi-byte characters.

Table 7–2: SQL-92 Language Elements That Support Internationalization and Localization *(9 of 11)*

Syntax	Description	Comment
REPLACE (*string_exp1* , *string_exp2* , *string_exp3*)	The scalar function REPLACE replaces all occurrences of *string_exp2* in *string_exp1* with *string_exp3*.	Each occurrence of *string_exp* and the result can contain multi-byte characters. Character comparisons are case-sensitive and depend on sort weights in the collation table in the database.
RIGHT (*string_exp* , *count*)	The scalar function RIGHT returns the rightmost count of characters of *string_exp*.	Each occurrence of *string_exp* and the result can contain multi-byte characters. Character comparisons are case-sensitive and depend on by sort weights in the collation table in the database.
RPAD (*char_expression* , *length* **[** , *pad_expression* **]**)	The scalar function RPAD pads the character string corresponding to the first argument on the right with the character string corresponding to the third argument. After the padding, the length of the result is equal to the value of the second argument *length*.	*char_expression* and *pad_expression* can contain multi-byte characters. *length* represents the number of characters in the result.

Table 7–2: SQL-92 Language Elements That Support Internationalization and Localization *(10 of 11)*

Syntax	Description	Comment
RTRIM (*char_expression* [, *char_set*])	The scalar function RTRIM removes all the trailing characters in *char_expression* that are present in *char_set* and returns the resultant string. The last character in the result is guaranteed not to be in *char_set*. If you do not specify a *char_set*, trailing blanks are removed.	The *char_expression*, the character set specified by *char_set*, and the result can contain multi-byte characters. Character comparisons are case-sensitive and depend on the collation table in the database.
SUBSTR (*char_expression* , *start_pos* [, *length*])	The scalar function SUBSTR returns the substring of the character string corresponding to the first argument starting at *start_pos* and *length* characters long. If the third argument *length* is not specified, the substring starting at *start_pos* up to the end of *char_expression* is returned.	*char_expression* and the result can contain multi-byte characters. *length* specifies a number of characters. Character comparisons are case-sensitive depend on sort weights in the collation table in the database.
SUFFIX (*char_expression* , *start_pos* , *char_expression*)	The scalar function SUFFIX returns the substring of a character string starting after the position specified by *start_pos* and the second *char_expression*, to the end of the string.	Each *char_expression* and the result can contain multi-byte characters. The *start_pos* argument specifies a character position, not a byte position. Character comparisons are case-sensitive depend on sort weights in the collation table in the database.

Table 7–2: SQL-92 Language Elements That Support Internationalization and Localization *(11 of 11)*

Syntax	Description	Comment
UCASE (*char_expression*)	The scalar function UCASE returns the result of the argument character expression after converting all the characters to uppercase. UCASE is identical to UPPER, but provides ODBC-compatible syntax.	A *char_expression* and the result can contain multi-byte characters. The conversion to uppercase depends on the case table in the convmap file. The default case table is BASIC.
UPPER (*char_expression*)	The scalar function UPPER returns the result of the argument character expression after converting all the characters to uppercase.	A *char_expression* and the result can contain multi-byte characters. The conversion to uppercase depends on the case table in the convmap file. The default case table is BASIC.

7.6.4 Format Specifiers Allowed With the TO_CHAR and TO_DATE Functions

The format specifiers allowed by that the SQL-92 TO_CHAR() and TO_DATE() functions appear in Table 7–3.

Table 7–3: Format Specifiers Allowed With TO_CHAR() and TO_DATE() *(1 of 4)*

Format Specifier	Description	Comment
AM	Meridian indicator in the native language Without periods	English only
A.M.	Meridian indicator in the native language With periods	English only
CC	The century as a two-digit number Computed as one greater than the first two digits of the four-digit year	Considers 1900 to be in the 20th century, 2000 to be in the 21st century, etc.
SCC	The century as a two-digit number. Computed as one greater than the first two digits of the four-digit year BC dates are prefixed by "-"	Considers 1900 to be in the 20th century, 2000 to be in the 21st century, etc.
D	The day of the week as a one-digit number between 1 and 7	–
DAY	The day of the week in the native language Entire name in uppercase	English only
Day	The day of the week in the native language First letter only in uppercase	English only

Table 7–3: Format Specifiers Allowed With TO_CHAR() and TO_DATE() *(2 of 4)*

Format Specifier	Description	Comment
day	The day of the week in the native language First letter only in uppercase	English only
DD	The day of the month as a two-digit number between 01 and 31	–
DDD	The day of the year as a three-digit number between 001 and 365	–
DY	The day of the week as a three-character string in the native language	English only
HH	The hour of the day as a two-digit number between 01 and 12	–
HH12	The hour of the day as a two-digit number between 01 and 12	Synonym of HH
HH24	The hour of the day as a two-digit number between 00 and 23	–
J	The Julian day	–
MI	The minute of the hour as a two-digit number between 0 and 59	–
MM	The month as a two-digit number between 01 and 12	–
MON	The first three characters of the name of the month in the native language	English only

Table 7–3: Format Specifiers Allowed With TO_CHAR() and TO_DATE() *(3 of 4)*

Format Specifier	Description	Comment
MONTH	The first nine characters of the name of the month in the native language, right-padded with blanks	English only
PM	Meridian indicator in the native language Without periods	English only
P.M.	Meridian indicator in the native language With periods	English only
Q	The quarter of the year as a single digit between 1 and 4	–
SS	Seconds as a two-digit number between 00 and 59	–
SSSS	Seconds past midnight as a number between 0 and 86339	–
TH	Ordinal suffix appended to a number Replaced by ST, ND, RD or TH depending on the last digit of the number	English only
W	The week of the month as a single digit between 1 and 5	–
WW	The week of the year as a two-digit number between 01 and 52	–
Y	The year as a single digit	–
YY	The year as a two-digit number	–

Table 7–3: Format Specifiers Allowed With TO_CHAR() and TO_DATE() *(4 of 4)*

Format Specifier	Description	Comment
YYY	The year as a three-digit number	–
YYYY	The year as a four-digit number	–
Y,YYY	The year as a four-digit number A comma separates the first digit from the other digits	–

8

Using Multi-byte Code Pages

Single-byte code pages can accommodate a maximum of 256 characters. This presents a problem for Chinese, Japanese, and Korean, whose ideographic and syllabic writing systems each contain tens of thousands of characters. This also presents a problem for Unicode, a standard that defines a code page to accommodate all characters of all languages of the world—including Chinese, Japanese, and Korean.

This chapter discusses a solution—multi-byte code pages—focusing on Chinese, Japanese, and Korean and touching on Unicode. For more information on Unicode, see Chapter 9, “Using Unicode.”

This chapter contains the following sections:

- Definitions For Key Terms
- Progress Support
- Input
- Output
- Inside the Multi-byte Application
- Issues Specific To Multi-byte Code Pages
- Guidelines For Using Multi-byte Characters

8.1 Definitions For Key Terms

This section defines technical terms that describe code pages, characters, and bytes.

8.1.1 Terms For Code Pages

The following terms describe code pages:

- A *single-byte* code page is one in which each character (or other symbol) has a numeric value expressible in a single byte.
- A *double-byte* code page is one in which each character (or other symbol) has a numeric value expressible in a maximum of two bytes.
- A *triple-byte* code page is one in which each character (or symbol) has a numeric value expressible in a maximum of three bytes.
- An *n-byte* code page is one in which each character (or symbol) has a numeric value expressible in a maximum of *n* bytes.
- A *multi-byte* code page is a code page that is not single byte.

8.1.2 Terms For Characters

The following terms describe characters:

- A *single-byte* character is one whose numeric value is expressible in one byte.
- A *double-byte* character is one whose numeric value is expressible in two bytes.
- A *triple-byte* character is one whose numeric value is expressible in three bytes.
- An *n-byte* character is one whose numeric value is expressible in exactly *n* bytes.
- A *multi-byte* character is one whose numeric value is not expressible in a single byte.

NOTE: A double-byte code page can contain single-byte characters and double-byte characters. A triple-byte code page can contain single-byte characters, double-byte characters, and triple-byte characters. And so on.

8.1.3 Terms For Bytes

The following terms describe bytes:

- A *lead byte* is first byte of a multi-byte character.
- A *trail byte* is any byte of a multi-byte character except the first.

8.1.4 Illustrations Of the Terms

Figure 8–1 and Figure 8–2 illustrate these terms.

Figure 8–1 shows four characters from a double-byte code page. The first character is single byte, the second and third characters are double byte, and the fourth character is single byte. Each double-byte character has a lead byte and a trail byte.

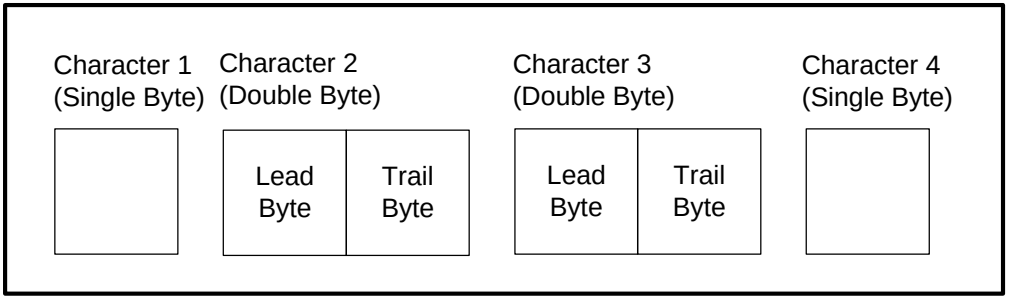

Figure 8–1: Four Characters From a Double-byte Code Page

Figure 8–2 shows three characters from a triple-byte code page. The first character is single byte, the second character is triple byte, and the third character is double byte. The triple-byte character has a lead byte and two trail bytes. The double-byte character has a lead byte and a trail byte.

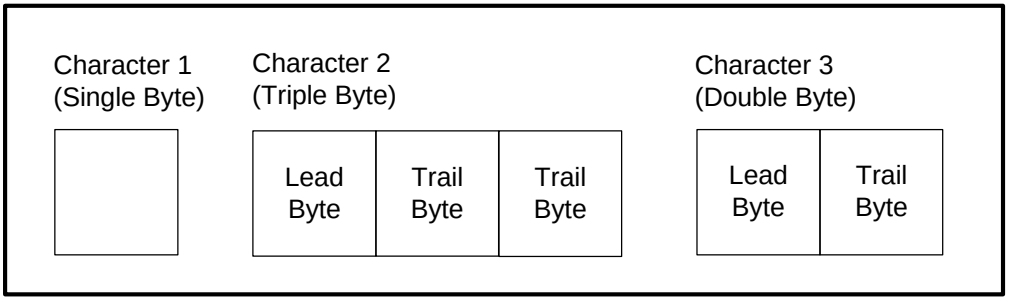

Figure 8–2: Three Characters From a Triple-byte Code Page

8.2 Progress Support

You can use multi-byte characters in every configuration that Progress supports, including client-server and batch. You are limited only by the multi-byte support provided by individual Progress products.

Multi-byte characters are supported by the majority of Progress products. Table 8–1 lists each Progress product and the multi-byte support it provides.

Table 8–1: Support For Multi-byte Characters *(1 of 3)*

Product	Multi-byte Support
AppBuilder	The AppBuilder—except for the Character Run window—supports double-byte characters. Widgets created with the AppBuilder can have double-byte characters in labels and in text. The Character Run window does not support double-byte or triple-byte characters.
AppServer	The AppServer supports double-byte and triple-byte characters.
Application Compiler	The Application Compiler supports double-byte and triple-byte characters. That is, the 4GL source code that the Application Compiler compiles can contain double-byte and triple-byte characters, and the resulting r-code supports double-byte and triple-byte characters.
Clients (graphical, character, and batch)	The graphical client and the Unix character client support double-byte characters but not triple-byte characters. The Windows character client does not support multi-byte characters. The batch client (started with the command `prowin32 -b`) supports multibyte.
Data Dictionary	The Progress Data Dictionary supports multi-byte characters.
DataServers	The DataServer for ORACLE supports double-byte characters if the ORACLE DBMS is set up for Native Language Support (NLS). For more information on NLS, see the ORACLE documentation. The DataServer for ORACLE does not support UTF–8.

Table 8–1: Support For Multi-byte Characters *(2 of 3)*

Product	Multi-byte Support
ESQL/C (SQL-89)	An ESQL/C program can have double-byte characters if the program's main procedure defines an external integer variable with the name "CharSet," and CharSet is assigned the name of the appropriate double-byte code page. Valid names, which reside in the file `sqlhli.h`, which PSC provides as part of the ESQL/C source, are: Japanese (128): SHIFTJIS_CHARSET or EUCJIS_CHARSET Korean (129): HANGEUL_CHARSET Simplified Chinese (134): GB2312_CHARSET Traditional Chinese (136): CHINESEBIG5_CHARSET Since Progress does not validate strings from ESQL/C programs, you must ensure that double-byte and triple-byte characters are not split.
HLC Applications	An HLC program can have double-byte and triple-byte characters. Since Progress does not validate strings from HLC programs, you must ensure that multi-byte characters are formed correctly and that they are not split. For example, Progress searches each character string for the NULL terminator. For each multi-byte character in the string, Progress searches only the lead byte. If the string ends in a multi-byte character and the NULL terminator resides (mistakenly) in a byte other than the position for a lead byte, Progress misses it and does not detect the end of the string correctly.
Procedure Editor	The Procedure Editor supports double-byte characters. To write a multi-byte Progress 4GL application, you can use any text editor that supports multi-byte characters. That is, you can use a multi-byte Unicode editor if you prefer.
Report Builder	The Report Builder does not support double-byte or triple-byte characters.

Table 8–1: Support For Multi-byte Characters *(3 of 3)*

Product	Multi-byte Support
SQL-89 Preprocessor	The SQL-89 Preprocessor supports multi-byte characters. Use the Language (-lng) startup parameter to specify the multi-byte code page for your program. Specify the -LANGUAGE option and the appropriate code-page name in the Preprocessor startup command. You can abbreviate the -LANGUAGE option as -L. It accepts these values: CP850, BIG5, GB2312, KSC5601, and SHIFTJIS. Note that the code-page names have no dashes (BIG5 and SHIFTJIS, not BIG-5 and SHIFT-JIS) when you use them in the Preprocessor startup command. For example, the following command starts the Preprocessor and specifies BIG–5, a code page for Traditional Chinese: `sqlcpp sqldemo.cc -L BIG5` If you do not specify the -LANGUAGE option, Progress uses the single-byte code page ibm850 as the default. This parameter setting is appropriate for all single-byte languages. Your C compiler must use the same code page as the one you specify with the -LANGUAGE option.
SQL-92	SQL-92 supports double-byte and triple-byte characters.
Translation Management System	The Translation Management System can manage translations to and from languages that use double-byte code pages.
WebClient	WebClient supports double-byte characters but not triple-byte characters.

8.3 Input

This section tells you how to get multi-byte characters into your application and includes the following subsections:

- Using the Keyboard and Mouse
- Using the 4GL
- Automatic Input Validation

8.3.1 Using the Keyboard and Mouse

To accommodate the large number of characters in Chinese, Japanese, and Korean, applications that use these languages typically use an input method editor. An *input method editor* (IME) is a program that accepts a keystroke combination, displays double-byte characters, and lets the user select one of them (by using the mouse or the keyboard). An *input method* maps a sequence of keystrokes to double-byte characters.

NOTE: On Windows, Progress does not provide its own input methods or IMEs. Rather, Progress supports the input methods and IMEs that Microsoft supplies and those that fully support the Microsoft standard.

Data Flow In Single-byte Applications

Using an IME affects the flow of input through Progress applications. In single-byte applications, which do not need IMEs, input data travels as follows:

1. From the user
2. Through the keyboard
3. Through the keyboard buffer
4. Through the Progress application
5. Through the Progress screen frame
6. To the monitor screen

Once the input data reaches the keyboard buffer, the programmer can query it using the 4GL's LASTKEY function and READKEY statement.

Figure 8–3 illustrates the flow of data through an application that does not use an input method editor.

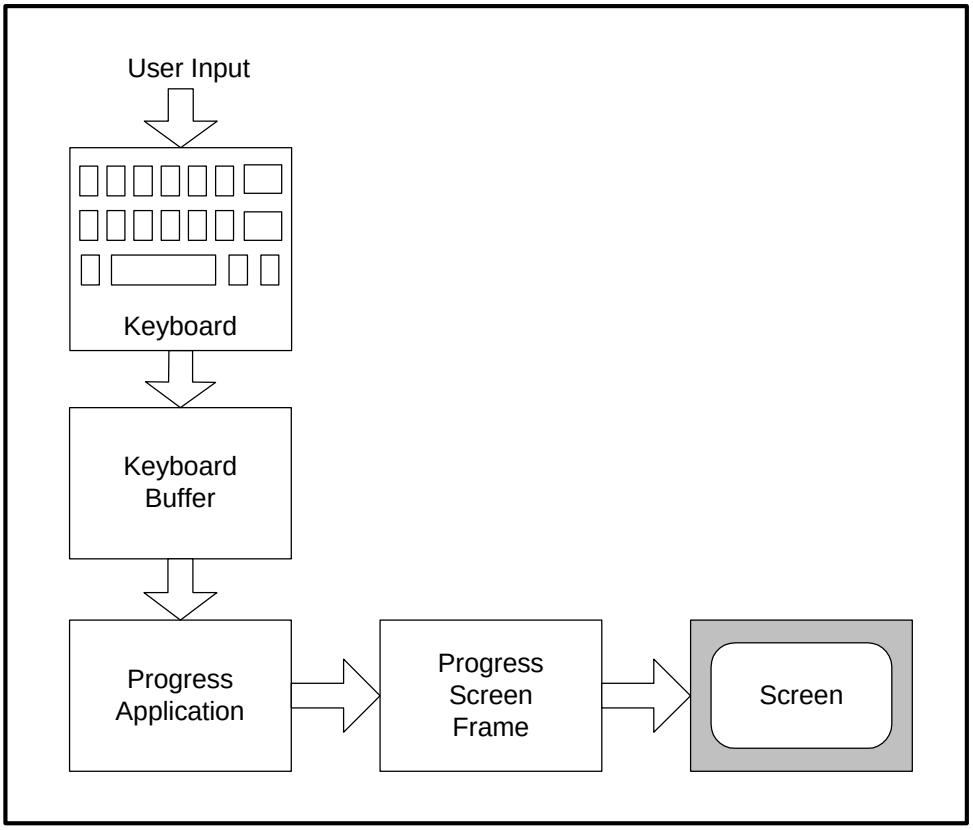

Figure 8–3: Application That Does Not Use an Input Method Editor

Data Flow In Double-byte Applications

In double-byte applications, which use IMEs, input data travels as follows:

1. From the user
2. Through the keyboard
3. Through the IME
4. Through the keyboard buffer
5. Though the Progress application
6. Through the Progress screen frame
7. To the screen

As before, once input data reaches the keyboard buffer, the programmer can query it using the 4GL's LASTKEY function and READKEY statement.

Figure 8–4 illustrates the flow of data through an application that uses an input method editor.

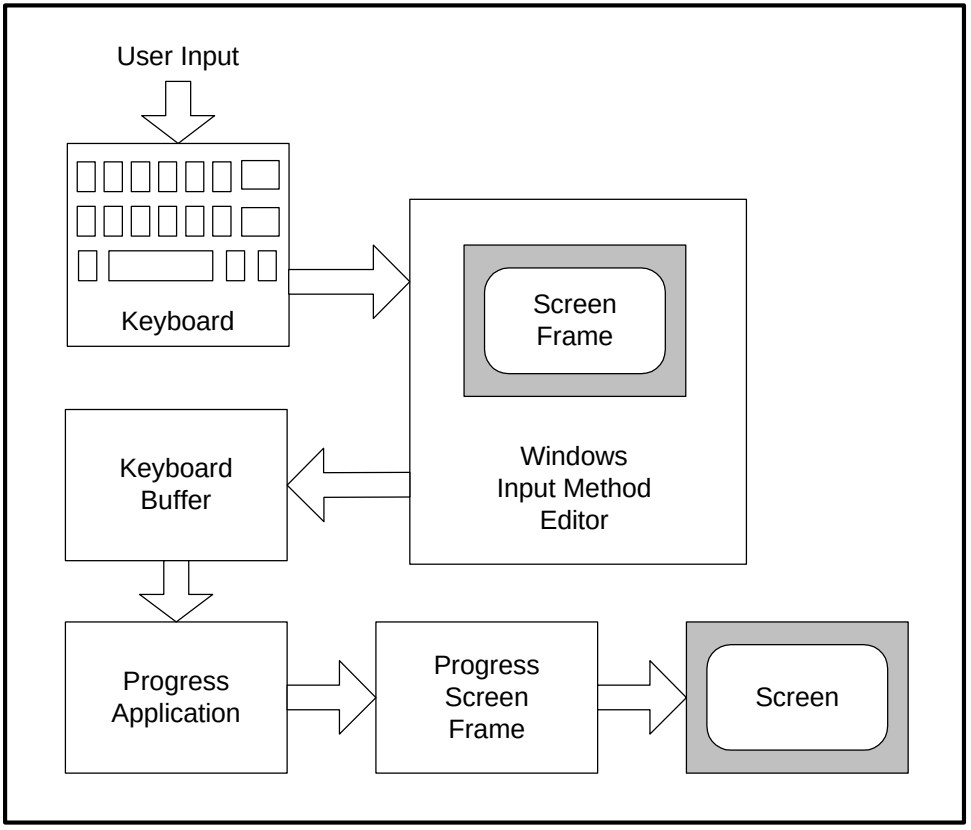

Figure 8–4: Application That Uses an Input Method Editor

8.3.2 Using the 4GL

Besides using the keyboard and possibly the mouse to enter multi-byte characters, you can also use the Progress 4GL. Specifically, this section covers:

- Using the 4GL to simulate inputting double-byte characters
- Using the 4GL to simulate pressing keys and clicking the mouse

Using the 4GL To Simulate Inputting Double-byte Characters

To input a double-byte character using the 4GL, assemble the character using the CHR function, whose syntax is:

SYNTAX

```
CHR ( expression
      [ , target-codepage [ , source-codepage ] ]
    )
```

The following example creates and displays the ⊟ double-byte character (which is from the KSC5601 code page, which supports Korean):

```
DEFINE VARIABLE Double-byte-char as CHAR.
Double-byte-character = CHR( (236 * 256) + 237 ).
DISPLAY Double-byte-char WITH 1 COLUMN.
```

In the preceding example, the second line shifts the first value into the lead byte and adds the second value to the trail byte.

NOTE: The preceding example assumes that the internal code page (-cpinternal) is set to KSC5601.

Using the 4GL to Simulate Key Presses and Mouse Clicks

Besides using the Progress 4GL to simulate the input of characters, including multi-byte characters, you can also use the 4GL to simulate key presses and mouse clicks in a multi-byte environment. Table 8–2 describes the 4GL elements involved and the multi-byte support provided by each.

Table 8–2: 4GL Elements That Simulate Key Presses and Mouse Clicks

4GL Element	Description	Example	Multi-byte Support
APPLY Statement	Applies an event to a widget or a procedure.	`APPLY "CHOOSE" TO order-amt IN FRAME x.`	The name of the event can contain multi-byte characters.
LASTKEY Function	Returns, as an integer, the key code of the most recent keyboard or mouse event LASTKEY returns values only after the input method places the data into the keyboard buffer.	`IF LASTKEY = KEYCODE("F9")...`	The key sequence can be a multi-byte character.
READKEY Statement	Reads one keystroke from an input source and sets the value of LASTKEY to the keystroke's keycode.	`READKEY.`	The key sequence can be a multi-byte character.

8.3.3 Automatic Input Validation

When you input multi-byte characters, Progress automatically validates the input under certain conditions. Validation of multi-byte characters means Progress checks that the lead byte has a value acceptable for lead bytes in this code page, and that each trail byte has a value acceptable for trail bytes in this code page.

CAUTION: Passing input validation is no guarantee that a multi-byte character is valid. Though its lead byte has an appropriate lead-byte value and each of its trail bytes have an appropriate trail-byte value, the combination of values might still be invalid for the character's code page.

Table 8–3 describes when validation does and does not occur.

Table 8–3: Automatic Validation Of Multi-byte Input

Data Source	Validation
Keyboard	Progress automatically checks that multi-byte characters are valid.
File	Progress does not check that multi-byte characters are valid, because Progress processes the data as bytes.
READKEY, SET, and UPDATE Statements	Progress automatically checks that multi-byte characters are valid, whether they came from the keyboard or from a file. Progress does not check that the lead-byte and trail-byte values constitute a valid character.
CHR Function	Progress automatically checks that multi-byte characters are valid, whether they came from the keyboard or from a file.

8.4 Output

Progress lets you use any font installed on your system that is appropriate for the language your application is running, whether that language uses a single-byte or multi-byte code page. For more information on setting the font, see Chapter 10, "Deployment and Configuration." The rest of this section discusses the Format phrase and printing.

8.4.1 The FORMAT Phrase

The Progress Data Dictionary Format field and the 4GL FORMAT phrase for character strings let you specify the display format in columns. For example, the character format "x(8)" can accommodate the following:

- Eight one-column characters

  NOTE: One-column characters exist in single-byte, double-byte, or triple-byte code pages.

- Four two-column characters

  NOTE: Two-column characters exist in double-byte and triple-byte code pages.

- Other combinations of one- and two-column characters totaling up to eight columns.

Fitting Multi-byte Characters Into the Available Columns

Before displaying or printing a multi-byte character, Progress ensures that the display or print field contains enough columns to accommodate the entire multi-byte character. If there are not enough columns, Progress does not display any of the multi-byte character, and pads the columns of the display field or print field with spaces.

NOTE: Progress neither splits nor truncates multi-byte characters to make them fit a display field.

For example, if you use the following format string, which specifies a seven-byte character field:

```
FORMAT "x(7)".
```

to display the following character string, which consists of four double-byte characters:

```
"日日日日"
```

Progress displays the first three double-byte characters in columns 1 through 6 of the field and a space in column 7, as shown below:

```
 "日日日  "
```

Progress does not display any of the fourth double-byte character, not even its lead byte, because doing so would split its lead byte from its trail byte, and the character would not appear properly.

Fitting Multi-byte Characters Around Formatting Characters

Before Progress displays a multi-byte character, if the next byte in the format specification is a formatting character, such as a dash, Progress inserts a space, then the dash, then searches the format specification for the next two available contiguous columns.

For example, if you use the following format, which consists of three bytes, a hyphen formatting character, and four bytes:

```
FORMAT "xxx-xxxx"
```

to display the following string, which consists of four double-byte characters:

```
"日中日中"
```

the result is the following string, which consists of the first double-byte character, a blank, the hyphen formatting symbol, the second double-byte character, and third double-byte character:

```
"日 -中日"
```

To build the resulting string, Progress:

1. Displays the first double-byte character.
2. Notices that only one column remains before the hyphen formatting character.
3. Inserts a blank in that column.
4. Displays the hyphen formatting character.
5. Displays the second double-byte character.
6. Displays the third double-byte character.
7. Notices that no more space remains.
8. Decides not to display the fourth double-byte character.

Allowing Single-byte Formatting Characters Only

Format strings cannot contain multi-byte characters. In the previous example, in the “xxx-xxxx“ format string, the x and the hyphen are single-byte characters. The following format string, which contains a double-byte character followed by two single-byte characters, is invalid:

```
FORMAT "日xx". /* invalid format string */
```

8.4.2 Printing

The information in the previous sections on displaying multi-byte characters also applies to printing multi-byte characters. Printing multi-byte characters, however, also involves the following issues:

- Printing from Windows
- Printing from character interfaces
- Testing printers

Printing From Windows

To print from a multi-byte Progress application running on Windows, whether the application’s interface is graphical or character, you can use any printer compatible with Windows and Windows device drivers.

For information on setting the font, see the *Progress Client Deployment Guide*.

Printing From Applications With a Character Interface

To print from a multi-byte Progress application with a character interface, make sure that the printer:

- Uses a font that can render each character that the application uses
- Uses a fixed-width (that is, monospaced) font
- Can print double-byte characters twice as wide as single-byte characters
- Uses the same code page as the application

Testing Printers

To see if your printer is configured correctly, create a test file. On the first line of the test file, enter the digits 0–9, repeating them as necessary to fill the line. On the second line of the test file, enter double-byte characters exclusively. After making sure that your printer is not set to draft mode, condensed mode, or proportional font mode, print the test file, either from the Windows character interface or from Progress. On the printed version of the test file you should see two single-byte characters above each double-byte character. If you do not see this, the printer is not configured correctly.

8.4.3 Character-client Color Limit

When you are using the Progress character client with a double-byte code page, the number of colors is limited to 31. To work around this:

1 ♦ Determine the 31 colors you want.

2 ♦ Make sure the 31 colors you want are specified as the first 31 colors in the `protermcap` file. For more information on the `protermcap` file, see the *Progress Client Deployment Guide*.

8.5 Inside the Multi-byte Application

Besides input code and output code, applications have internal code. The primary responsibility of the internal code is to process characters and character strings, which, of course, can be multi-byte. This section discusses:

- Distinguishing characters, bytes, and columns
- Techniques for working with multi-byte characters
- 4GL elements that support the processing of multi-byte characters

8.5.1 Distinguishing Characters, Bytes, and Columns

In order for your code to process multi-byte characters correctly, you need to understand the difference between characters, columns, and bytes:

- *Characters* are symbols in a code page, each of which is assigned a numeric value.
- *Bytes* are units of storage, each consisting of eight bits.
- *Columns* are units of width, indicating how much width a symbol requires on the monitor or on a printed report.

To clarify these concepts, examine Table 8–4, which shows two characters, one single-byte and one double-byte, and the byte count and column count of each.

Table 8–4: Byte Count and Column Count

Character	Number of Bytes	Number of Columns
A	1	1
中	2	2

8.5.2 Techniques For Working With Multi-byte Characters

The following techniques might save you time and trouble:

- Choosing the appropriate unit of measure
- Testing a character string for the presence of multi-byte characters
- Testing a byte for a lead-byte value

Choosing the Appropriate Unit Of Measure

Several Progress 4GL elements, including the LENGTH function, the OVERLAY statement, the SUBSTRING function, and the SUBSTRING statement, let you specify the unit of measure as the character, the byte, or the column. If you choose the wrong unit of measure, you might split or overlay a multi-byte character. Consider the following example:

```
DEFINE VARIABLE char-over AS CHARACTER FORMAT "X(8)".
char-over = "abc中efg".
OVERLAY(char-over),1,4,"RAW") = "wxyz". /* RAW is wrong */
DISPLAY char-over WITH 1 COLUMN.
```

The example defines a character variable and sets it to a string of seven characters, the fourth of which is double byte. The example then overlays a string of four characters, all single byte, on the original string, starting at position one and continuing for four positions. Unfortunately, the unit of measure is the byte (specified by RAW), so the fourth byte of the second string, which is the character z, overlays the fourth byte of the original string, which is the lead-byte of the double-byte character.

Figure 8–5 shows how the z in the second string overlays the lead byte of the double-byte character in the original string.

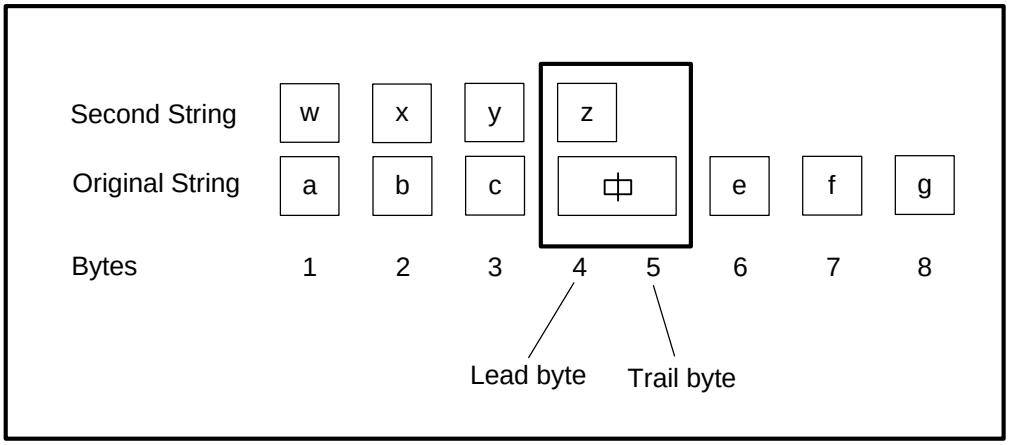

Figure 8–5: A Single-byte Character Overlaying a Lead Byte

All that remains of the multi-byte character is the trail-byte, as shown in Figure 8–6.

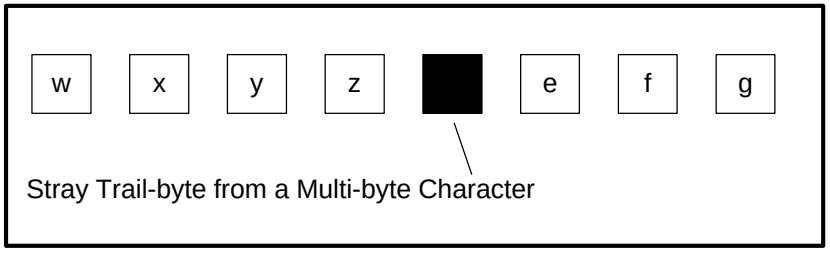

Figure 8–6: Result Of a Single-byte Character Overlaying a Lead Byte

To fix this error, change the unit of measure to characters. The corrected program is as follows:

```
DEFINE VARIABLE char-over AS CHARACTER FORMAT "X(8)".
char-over = "abc□□efg".
OVERLAY(char-over),1,4,"CHARACTER") = "wxyz". /* CHARACTER is wrong */
DISPLAY char-over WITH 1 COLUMN.
```

The corrected program produces the string shown in Figure 8–7.

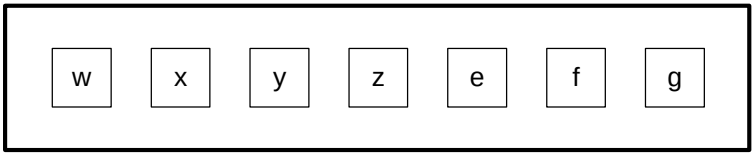

Figure 8–7: String Produced By an OVERLAY Statement Whose Unit Of Measure Is the Character

Testing Character Strings For Multi-byte Characters

To determine whether a character string contains multi-byte characters, use the LENGTH function, which returns the number of characters, bytes, or columns in a string. The syntax is:

SYNTAX

```
LENGTH ( { string [ type ] | raw-expression } )
```

string

A character expression. The specified *string* can contain double-byte characters.

type

A character expression that indicates whether you want the length of a string in character units, bytes, or columns. A double-byte character registers as one character unit. By default unit of measurement is character units.

There are three valid types: CHARACTER, RAW, and COLUMN. The expression "CHARACTER" indicates that the length is measured in characters, including double-byte characters. The expression "RAW" indicates that the length is measured in bytes. The expression "COLUMN" indicates that the length is measured in columns. If you specify the *type* as a constant expression, Progress validates the type specification at compile time. If you specify the *type* as a variable expression, Progress validates the type specification at run time.

raw-expression

A function or variable name that returns a raw value.

To use the technique, call LENGTH twice: once with the CHARACTER option, which returns the length in characters, and once with the RAW option, which returns the length in bytes. Then, compare the two lengths. If they are equal, the string contains only single-byte characters. Else, the string contains at least one multi-byte character.

The following examples illustrate the technique:

The first example tests a character string consisting of one double-byte character. Since the length of the string in characters (1) does not match the length in bytes (2), the example displays `Multi-byte characters in the string:`

```
DEFINE VARIABLE mychar AS CHARACTER INITIAL "⊟".
IF LENGTH (mychar, "CHARACTER") = LENGTH(mychar, "RAW")
THEN DISPLAY "No multi-byte character in the string".
ELSE DISPLAY "Multi-byte characters in the string".
```

The second example tests a character string consisting of three single-byte characters. Since the length of the string in characters (3) matches the length in bytes (3), this example displays `No multi-byte characters in the string.`

```
DEFINE VARIABLE mychar AS CHARACTER INITIAL "123".
IF LENGTH(mychar,"CHARACTER") = LENGTH(mychar,"RAW")
THEN DISPLAY "No multi-byte characters in the string".
ELSE DISPLAY "Multi-byte characters in the string".
```

Testing For a Lead-Byte Value

The next technique involves testing a byte for a lead-byte value. Lead bytes (and trail bytes) often have special values to distinguish them. Table 8–5 lists the lead-byte and trail-byte values for the multi-byte code pages that Progress supports.

Table 8–5: Lead Byte and Trail Byte Values

Code Page	Language or Standard	Lead-byte Values	Trail-byte Values
BIG–5	Traditional Chinese	161 through 254	64 through 126 161 through 254
CP949	Korean	129 through 254	65 through 90 97 through 122 129 through 254
CP950	Traditional Chinese	129 through 254	64 through 126 128 through 254
CP1361	Korean	132 through 211 216 through 222 224 through 249	65 through 127 129 through 254
EUCJIS	Japanese	142 164 through 254	161 through 254
GB2312	Simplified Chinese	161 through 254	161 through 254
KSC5601	Korean	161 through 254	161 through 254
SHIFT–JIS	Japanese	129 through 159 224 through 252	64 through 126 128 through 252
UTF–8	Unicode	193 through 239	128 through 191

NOTE: You cannot always assume that a byte with a lead-byte value is a lead byte, or that a byte with a trail-byte value is a trail byte. This is because the possible values for trail bytes overlap those of lead bytes and single bytes. For example, the value 164 can correspond to a lead byte or to a trail byte. To determine which it is, you must inspect the string.

To determine if a byte has a lead-byte value, use the IS–LEAD–BYTE function, which evaluates a character expression and returns TRUE if the first byte of the first character of the character string has a value within the range permitted for lead bytes. Otherwise, IS–LEAD–BYTE returns FALSE. IS–LEAD–BYTE has the following syntax:

SYNTAX

```
IS–LEAD–BYTE ( string )
```

string

A character expression (a constant, field name, variable name, or any combination of these) whose value is a character.

In the following example, IS–LEAD–BYTE examines a string whose first character is single byte. Since the first byte of the first character of the string is not a lead byte, its value is not within the range permitted for lead bytes, IS–LEAD–BYTE returns FALSE, and the example displays `Lead: no`:

```
DEFINE VARIABLE lead AS LOGICAL.
lead = IS-LEAD-BYTE("xy中").
DISPLAY lead WITH 1 COLUMN.
```

The following example is identical to the preceding example except that the first character of the string is double byte. Since the first byte of the first character of the string is a lead byte, its value falls within the range permitted for lead bytes, IS–LEAD–BYTE returns TRUE, and the example displays `Lead: Yes`:

```
DEFINE VARIABLE lead AS LOGICAL.
lead = IS-LEAD-BYTE("中xy").
DISPLAY lead WITH 1 COLUMN.
```

8.5.3 4GL Support For Processing Multi-byte Characters

The Progress 4GL supports multi-byte characters in general and has many elements that support multi-byte characters in particular. Table 8–6 describes these language elements.

Table 8–6: 4GL Elements For Processing Multi-byte Applications *(1 of 3)*

4GL Element	Brief Description	Multi-byte Support
ASC Function	Returns the integer that corresponds to a given character.	You can specify source and target code pages, which can be multi-byte. For a double-byte character, ASC returns a value greater than 255 and less than 65536. For a triple-byte character, ASC returns a value less than 15,712,191 (hex 0xEFBFBF). For an invalid character, ASC returns -1.
CHR Function	Returns the character that corresponds to a given integer in the target code page.	You can specify source and target code pages. If you do, CHR converts the value from source to target before returning it to you. Code pages can be multi-byte. If the INTEGER value is greater than 255 and less than 15,712,191, CHR checks for a valid lead-byte value. If found, CHR returns a double-byte character. If the INTEGER expression exceeds 15,712,191 or does not correspond to a valid lead-byte or trail-byte value, CHR returns a null string.

Table 8–6: 4GL Elements For Processing Multi-byte Applications *(2 of 3)*

4GL Element	Brief Description	Multi-byte Support
ENCODE Function	Encodes a character string.	The character string can contain multi-byte characters.
FORMAT Phrase	Specifies the display format of, among other things, character strings.	For a field that contains multi-byte characters, you can specify the width in columns.
INDEX Function	Returns the position of one character string within another.	The character strings can contain multi-byte characters.
IS–LEAD–BYTE Function	Returns TRUE if the first byte of a character string has a value that falls within the range permitted for lead bytes.	Expressly designed for multi-byte characters.
LENGTH Function	Returns the number of characters, bytes, or columns in a character string.	Characters can be multi-byte. You can set the unit of measure to characters, bytes, or columns.
OVERLAY Statement	Overlays a character string on a variable or field at the given position and optionally for the given length.	Characters can be double byte or triple byte. You can set the unit of measure to characters, bytes, or columns.
STRING Function	Converts a value of any data type to CHARACTER.	The value can contain double-byte or triple-byte data

Table 8–6: 4GL Elements For Processing Multi-byte Applications *(3 of 3)*

4GL Element	Brief Description	Multi-byte Support
SUBSTRING Function	Returns a specified portion of a character string.	Characters can be multi-byte. You can set the unit of measure to characters, bytes, or columns.
SUBSTRING Statement	Overlays a specified portion of a character string with another character string.	Characters can be multi-byte. You can set the unit of measure to characters, bytes, or columns.

8.6 Issues Specific To Multi-byte Code Pages

Issues specific to multi-byte code pages are discussed in the following sections:

- Progress Support For Multi-byte Code Pages
- Valid and Invalid Code-page Conversions
- User-defined Characters
- Collating Multi-byte Characters
- Default Word-break Behavior Of Characters In Multi-byte Code Pages

8.6.1 Progress Support For Multi-byte Code Pages

Table 8–7 lists the multi-byte code pages that Progress supports.

Table 8–7: Multi-byte Code Pages That Progress Supports

Code Page	Language
BIG–5	Traditional Chinese
CP949	Korean
CP950	Traditional Chinese
CP1361	Korean
EUCJIS	Japanese
GB2312	Simplified Chinese
KSC5601	Korean
SHIFT–JIS	Japanese
UTF–8	Multilingual

8.6.2 Valid and Invalid Code-page Conversions

Sometimes you can convert data from one code page to another and sometimes you cannot. Chapter 2, "Understanding Code Pages," discusses a general rule, based on elementary set theory, that explains why some code-page conversions are valid and others are not. In the following restatement of the rule, *source code page* means the code page you are converting from and *target code page* means the code page you are converting to.

Determining Valid Code-page Conversions (For Non-unicode Databases)

You can convert data from one code page to another if one of the following conditions is true:

- Each symbol of the source code page appears in the target code page.
- Each symbol that appears in the data appears in the target code page.

If Progress is converting from UTF-8 and encounters a character that does not exist in the target code page, Progress substitutes the question mark (?).

Converting Between Double Byte and Single Byte

If you apply the preceding rule to conversion between a double-byte code page and a single-byte code page, you conclude that all such conversions are invalid, in either direction. This is because double-byte code pages contain many more symbols than single-byte code pages.

Converting Between Double Byte and Double Byte

If you apply the preceding rule to conversion from one double-byte code page to another, you conclude that a conversion is valid if the source code page is a subset of the target code page. The double-byte to double-byte conversions that appear in Table 8–8 are valid.

Table 8–8: Valid Double-byte To Double-byte Code-page Conversions

Conversion	Comment
SHIFT–JIS to EUCJIS EUCJIS to SHIFT–JIS	SHIFT–JIS and EUCJIS contain the same symbols.
KSC5601 to CP949	KSC5601 is a subset of CP949.
BIG–5 to CP950	BIG–5 is a subset of CP950.

NOTE: You cannot convert from CP949 and KSC5601, both code pages for Korean, because CP949 contains many characters that KSC5601 does not.

8.6.3 User-defined Characters

Progress supports user-defined characters, such as the Japanese *Gaiji* (external) characters of the SHIFT–JIS code page. The lead bytes of these characters are distinguished by having values in the range 240 through 252. The trail bytes of these characters have values in the same range as any SHIFT–JIS trail byte—namely, 64 through 126 and 128 through 252.

Limitations Of User-defined Characters

Precisely because these characters are user defined, administrators must consider carefully whether to include them in a database. If user-defined characters occur in a database, all systems that access the database must define them consistently. Also, there are no standard algorithms for converting user-defined characters from one code page to another. For example, Progress cannot convert user-defined characters from SHIFT-JIS to EUCJIS, since a user-defined character appearing in multiple code pages does not necessarily occupy the corresponding position in each code page.

NOTE: You cannot use strings containing user-defined characters with the CODEPAGE-CONVERT() function. Instead, use the REPLACE() function.

Guidelines For Using User-defined Characters

CAUTION: If a Progress application has a tier (client, server, etc.) built earlier than Version 9.0B or 9.1A, you must alter the lead-byte table to include the lead-bytes of the Gaiji characters.

When you use user-defined characters, do not forget to:

- Go into the SHIFT–JIS `convmap` file and, in the lead-byte attribute table, set the value of each byte in the range 240 through 252 (or perhaps only some of them, depending on how many user-defined characters you want) to 1, to indicate a lead byte.

  The SHIFT–JIS CONVMAP file is called `shiftjis.dat` and resides in the `DLC/prolang/convmap` directory. For more information on CONVMAP files, see Chapter 3, “Understanding Character Processing Tables.”

- Start each database server with the -cpinternal startup parameter set to SHIFT-JIS.
- Start each client with the -cpinternal startup parameter set to SHIFT-JIS.
- Connect to a SHIFT–JIS database.

8.6.4 Collating Multi-byte Characters

When you sort multi-byte characters, you face a question that you do not face when sorting single-byte characters: in what order should the different types of character sort? That is, should all one-byte characters sort before all two-byte characters? Should all two-byte characters sort before all three-byte characters? And how should the user-defined characters of the SHIFT–JIS code page sort?

The default collation table that Progress provides for the double-byte Asian languages (Chinese, Japanese, and Korean) sorts all single-byte characters before all double-byte characters. Table 8–9 shows how Progress sorts Japanese characters.

Table 8–9: Japanese Collation Order By Character Type

Character Type	Range of Values
Single Byte (ASCII)	0–127
Single Byte (half-width Katakana)	160–223
Lead Byte (range 1)	129–159
Lead Byte (range 2)	224–239
User-defined (Gaiji)	240–252

NOTE: You can modify the sort order of lead bytes, though not the sort order of trail bytes. For more information on modifying the sort order of lead bytes, see the comments in the BASIC collation table for the SHIFT–JIS code page in the `japanese.dat` file in the `DLC/prolang/convmap` directory.

Sort Order Of Trail Bytes

For a given lead byte, trail bytes sort in binary order. For example, if one double-byte character has a lead-byte value of 159 and a trail-byte value of 100, and another double-byte character has a lead-byte value of 159 and a trail-byte value of 170, the character with byte values 159 and 100 sorts before the character with byte values 159 and 170. Figure 8–8 illustrates this.

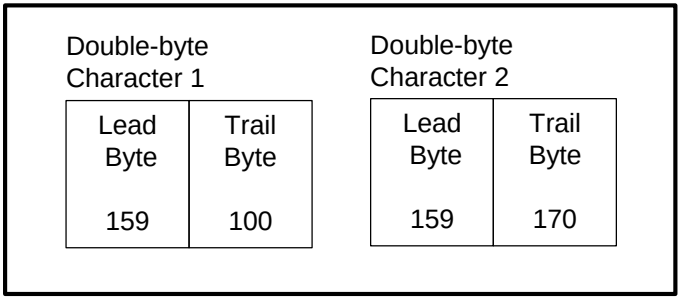

Figure 8–8: Sorting Double-byte Characters

8.6.5 Default Word-break Behavior Of Characters In Multi-byte Code Pages

Table 8–10 describes the default word-break behavior of characters in multi-byte code pages.

NOTE: Table 8–10 assumes that word-break tables are Version 9 Type 3. For more information on word-break tables, see the "Word-break Tables" section in Chapter 3, "Understanding Character Processing Tables."

Table 8–10: Default Word-break Behavior Of Characters In Multi-byte Code Pages

If the Code Page Is...	And the Characters Are...	The Characters Behave (By Default)...
Double byte	Single Byte	Depending on whether they are alphabetic or nonalphabetic. This is specified in the code page's character-attribute table. To change the default word-break behavior, supply a word-break table input file.
Double byte	Double Byte	As separate words.
UTF-8	Single Byte	Depending on whether they are alphabetic or nonalphabetic. This is specified in the code page's character-attribute table. To change the default word-break behavior, supply a word-break table input file.
UTF-8	Two-byte UTF-8	Corresponding to the USE_IT word-delimiter attribute.
UTF-8	Three-byte UTF-8	As separate words.

NOTE: The default word-break behavior can be changed only for single-byte characters.

For more information on character-attribute tables, see the "Character Attribute Tables" section. For more information on modifying word-break tables, see the "Creating and Modifying Word-break Tables" section. For more information on word-delimiter attributes, see the "Understanding Word-delimiter Attributes" section. All these sections resides in Chapter 3, "Understanding Character Processing Tables."

8.7 Guidelines For Using Multi-byte Characters

When you use multi-byte characters in Progress applications, the following guidelines apply:

- When choosing a Chinese, Japanese, or Korean font, choose one large enough to display each character cleanly and clearly.
- When designing an application that uses Chinese, Japanese, or Korean, leave sufficient space in the user interface for the IME.
- When using multi-byte characters, when using a 4GL element to calculate the length of a character string or a position within a character string, use the correct unit of measure, whether bytes, characters, or columns.

  For more information on how to specify bytes, characters, or columns, see the 4GL element's reference entry in the *Progress Language Reference*.

  To have the compiler warn you of 4GL elements in the source code that should specify the unit of measure and that appear not to, start Progress with the Check Double-byte Enabled (-checkdbe) startup parameter. For more information on -checkdbe, see the *Progress Startup Command and Parameter Reference*.
- When displaying or printing characters, do not assume that one character requires one column.

- When assigning an accelerator to a menu item or label, use a single-byte character. If you use a multi-byte character, Windows underlines only the first byte.

  For example, the following code fragment assigns the accelerator “A” to a button whose label is a double-byte character:

```
DEFINE BUTTON btn-Test LABEL "中 (&A)".
```

- When you use the UNIX character client with double-byte code pages, UNIX strips input characters to seven bits, which might garble data. To avoid this, before starting Progress, enter the following command at the command prompt:

```
stty -istrip
```

9

Using Unicode

When you adapt an application to a different locale, the character set often changes. One way to deal with multiple character sets is to use multiple code pages, one for each language or group of languages. However, this approach forces you to deal with code-page conversion and code-page incompatibility, explained in Chapter 2, “Understanding Code Pages” and Chapter 8, “Using Multi-byte Code Pages.” Another approach is to use a single code page that includes all characters of all languages of the world. This is the idea behind Unicode, which is the focus of this chapter.

This chapter includes the following sections:

- What Is Unicode
- Why Use Unicode
- Using Unicode With Progress Products
- Using Unicode With Progress Databases
- Using Unicode With Progress Applications
- Guidelines For Using Unicode

9.1 What Is Unicode

An evolving standard, Unicode defines a single code page that includes most symbols—letters, ideograms, syllabics (such as the Japanese Kana symbols), punctuation, diacritics, mathematical symbols, technical symbols, etc.—from most of the languages of the world, and assigns each symbol a numeric value—originally, a number between zero and 65,535, the range of an unsigned 16-bit integer.

As it turned out, Unicode's original limit of 65,536 symbols proved too small, and the limit was extended to well over 1,000,000 symbols. Several ways of encoding each symbol were defined, and the encodings were designed so that you can convert from one to another any number of times without losing any information. For more information on the algorithms for converting between encodings, see the Unicode web site, `www.unicode.org`. Progress supports Unicode's UTF–8 encoding.

9.2 Why Use Unicode

The Unicode approach, with its UTF–8 encoding and multi-byte characters, might seem complicated. But the other approach, using multiple code pages, can be even more complicated.

9.2.1 The Limits Of Multiple Code Pages

Applications that use multiple code pages are often difficult to design, deploy, configure, and run, for the following reasons:

- You must often convert data from one code page to another.
- Before you perform a code page conversion, you must determine whether the source and target code pages are compatible.
- You must get application components that use incompatible code pages to read, write, and display each other's data, which can be difficult or impossible.

  This applies especially to applications that read and write multilingual data to a database and then display the data on client monitors and on printed reports.

9.2.2 The Advantages Of Unicode

Unicode has many advantages. Table 9–1 lists them.

Table 9–1: Unicode's Advantages *(1 of 2)*

Advantage	Explanation
Simplified application development	When an application component uses Unicode, all symbols needed by the application for reading and writing character data reside in a single code page. This simplifies application development enormously.
Ease of migration of existing code	UTF–8 includes the traditional ASCII characters in its first 127 positions and assigns each of these characters its traditional ASCII value. This simplifies adapting existing ASCII applications to Unicode.
Ease of providing shared access to data	Progress clients that use incompatible code pages can easily read and write a single UTF–8 database. Progress automatically converts the code page as data passes between the client and the database.
Ease of worldwide deployment	UTF–8 databases and r-code files are multilingual. They can be deployed worldwide.
Interoperability	Active-X and Java clients are Unicode based. They can communicate with UTF–8 databases and application servers.

Table 9–1: Unicode's Advantages *(2 of 2)*

Advantage	Explanation
Web compatibility	Unicode is becoming the universal code page of the Web. Current Web standards require Unicode and rely on it.
Multilingual applications	Applications using Unicode can support multiple languages: • In the data • In the user interface • In reports

9.3 Using Unicode With Progress Products

Some Progress products can run directly in Unicode (that is, with -cpinternal utf-8). Other Progress products cannot, but can run in a code page that Progress can convert to and from Unicode.

The following products can run in Unicode:

- ActiveX client
- AppServer
- Batch client
- Database server
- 4GL compiler and r-code
- Java client
- SQL-92

9.4 Using Unicode With Progress Databases

Using UTF-8 with Progress databases involves several techniques, covered in the following sections:

- Converting a Progress Database To UTF-8 Using the PROUTIL CONVCHAR Utility
- Converting a Progress Database To UTF-8 Using Dump and Load
- Compiling, Storing, and Applying the UTF-8 Word-break Rules To a Database

9.4.1 Converting a Progress Database To UTF-8 Using the PROUTIL CONVCHAR Utility

To convert a Progress database to UTF-8 using the PROUTIL CONVCHAR utility:

CAUTION: Before you begin, back up your database.

1 ♦ Convert the database to UTF-8 using the following syntax:

SYNTAX

```
proutil database-name -C convchar convert utf-8
```

2 ♦ Load the UTF-8 collation table using the syntax for your operating system:

Windows Syntax	UNIX Syntax
%DLC%\prolang\utf_tran.df	$DLC/prolang/utf/_tran.df

3 ♦ Compile, store, and apply the UTF-8 word-break rules to your database. For complete instructions, see the "Compiling, Storing, and Applying the UTF-8 Word-break Rules To a Database" section in this manual.

4 ♦ Rebuild the indexes using the following syntax:

SYNTAX

```
proutil database-name -C idxbuild
```

9.4.2 Converting a Progress Database To UTF-8 Using Dump and Load

To convert a Progress database to UTF-8 using the DUMP and LOAD utilities:

CAUTION: Before you begin, back up your database.

1 ♦ Dump the schema and data of the existing database using the Data Administration utility. (From the Procedure Editor main menu, select **Tools→ Data Administration→ Admin→ Dump Data and Definitions**.)

2 ♦ Create a new, empty UTF-8 database using the syntax for your operating system:

Windows Syntax	UNIX Syntax
`prodb` *`database-name`* `%DLC%\prolang\utf\empty.db`	`prodb` *`database-name`* `$DLC/prolang/utf/empty.db`

3 ♦ Compile, store, and apply the UTF-8 word-break rules to the database. For complete instructions, see the "Compiling, Storing, and Applying the UTF-8 Word-break Rules To a Database" section in this manual.

4 ♦ Load the schema and data to the database using the Data Administration utility. (From the Procedure Editor main menu, select **Tools→ Data Administration→ Admin→ Load Data and Definitions**.)

NOTE: When the data are loaded, the indexes are automatically rebuilt.

9.4.3 Compiling, Storing, and Applying the UTF-8 Word-break Rules To a Database

When you convert an existing database to UTF-8, whether you use the PROUTIL CONVCHAR utility or the DUMP and LOAD utilities, you must compile, store, and apply the UTF-8 word-break rules to the database.

If you forget to apply the word-break rules to your database, you might get the following symptoms:

- Queries with the CONTAINS operator return incorrect results.
- You get a QBW syntax error sayning that an asterisk (*) is allowed only at the end of a word.

To compile, store, and apply the UTF-8 word-break rules to a database, following these steps:

1 ♦ Compile a new version of the word-break table for UTF-8 using the syntax for your operating system:

Windows Syntax	UNIX Syntax
`proutil -C wbreak-compiler %DLC%\prolang\convmap\utf-8.bas` *`number`*	`proutil -C wbreak-compiler $DLC/prolang/convmap/utf-8.bas` *`number`*

where *`number`* indicates an INTEGER between 1 and 255.

This produces a new word-break table, `proword.`*`number`*.

2 ♦ Store the new word-break table using one of the following techniques:

- Store it in the `$DLC` directory on UNIX or in the `%DLC%` directory on Windows.
- Store it in an arbitrary directory, then set the environment variable PROWD*`number`* to the value of the arbitrary directory.

3 ♦ Apply the new word-break rules to the database using the following syntax:

SYNTAX

```
proutil database-name -C word-rules number
```

9.5 Using Unicode With Progress Applications

Creating and running an application that uses Unicode is not difficult. Here is an example of creating and running an application consisting of a database server, a graphical client, and a UTF–8 database.

Since UTF–8 is supported by the Progress database server but not by the Progress graphical or character client:

- Each graphical or character client must start up in a code page other than UTF–8.
- The programmer must ensure that a graphical or character client accesses only records in a compatible code page.
- The programmer must follow guidelines for multi-byte programming, such as distinguishing characters, bytes, and columns.

To create the application:

1 ♦ Convert the database to Unicode using one of the techniques in the "Using Unicode With Progress Databases" section.

2 ♦ Design queries that access only records that use client's code page.

One way to do this is for tables to have a field that indicates the record's code page. When records are added, the field is populated. When the database is queried, the query references the code page field in order to return only those records in the client's code page.

Imagine that in the Sports database, the Customer table has a field, `db-language`, indicating the code page or language of the record. A client whose language corresponds to the value of the variable `user-language` might submit a query similar to the following:

```
FOR EACH customer WHERE db-language = user-language:
  DISPLAY name address city country comments.
END.
```

3 ♦ Start a Progress database server, setting the server's code page to UTF–8.

The following command fragment illustrates this:

```
proserve -cpinternal utf-8 -cpstream ...
```

4 ♦ Start a client in the native code page (perhaps ISO8859–15). Set -cpinternal, -cpstream, and the other code-page-related startup parameters to this code page.

The following command illustrates this:

```
prowin32 -cpinternal iso8859-15 -cpstream iso8859-15
```

Your Unicode application is up and running.

9.6 Guidelines For Using Unicode

When you use Unicode in Progress applications, the following restrictions, cautions, and suggestions apply:

- To encode a single character consisting of a base character below and a diacritic above, such as the Spanish ñ, data must use the *composed character*—that is, the single encoding that represents the character and the diacritic. If the data encodes the base character and the diacritic separately, Progress does not interpret these as a single character consisting of the base character below and the diacritic above.

- Progress sorts UTF–8 data in binary order. If an application requires sorting in a different order, the application must provide the sort logic itself.

  NOTE: The sort logic might use the COLLATE option of the FOR statement, the OPEN QUERY statement, and the PRESELECT phrase. For more information on this 4GL element, see the *Progress Language Reference*.

- Progress supports code page conversion to and from UTF–8 the same way it supports code page conversion to and from other code pages. For more information on code page conversion, see Chapter 2, "Understanding Code Pages," and Chapter 3, "Understanding Character Processing Tables."

- When an existing database is converted to UTF–8, the amount of storage required by each non-ASCII character increases. Roughly, each non-ASCII Latin-alphabet character converted to UTF–8 tends to require two bytes, while each double-byte Chinese, Japanese, or Korean character converted to UTF–8 tends to require three bytes.

- To display and print Unicode data, consider using a Unicode font. They are available commercially.

10

Deployment and Configuration

Progress supplies a number of configuration files — specifically, initialization files, message files, parameter files, and property files — that are especially useful for localizing applications. Some of these files you modify directly. Others you modify by using the Progress Explorer. Also, many Progress products provide parameters that let you localize the output.

This chapter includes the following sections:

- The progress.ini File and the Windows Registry
- The PROTERMCAP File
- The PROMSGS File
- Parameter Files
- The conmgr.properties File
- Additional Guidelines

NOTE: Many of the files and parameters described in this chapter depend on the CONVMAP file, which contains character attribute tables, case conversion tables, and code page conversion tables, all for a particular code page. For more information on the CONVMAP file, these tables, and code pages, see Chapter 2, “Understanding Code Pages,” and Chapter 3, “Understanding Character Processing Tables.”

10.1 The progress.ini File and the Windows Registry

The `progress.ini` file specifies elements of the Windows user interface—for example, colors and fonts—that often vary across locales. Tailoring the `progress.ini` file is an important part of adapting an application to a particular locale.

Progress supports the Windows registry and searches the registry first for system configuration information. However, you can still use initialization (`.ini`) files to ensure that applications are deployed and configured correctly and consistently across customer sites. The Progress installation program automatically copies information from the initialization file to the registry during installation. After installation, you can modify the initialization file, then run the Progress INI2REG utility to copy the new information from the initialization file to the registry. For information on the registry, on maintaining system configuration information, and on the INI2REG utility, see the *Progress Client Deployment Guide* and the *Progress Installation and Configuration Guide Version 9 for Windows*.

NOTE: The `progress.ini` file and the registry are used by Windows graphical-interface and character-interface applications only. They are not used by UNIX character-interface applications.

The sections of the `progress.ini` file that typically have settings you might want to localize are the [Startup] section and the [fonts] section.

10.1.1 [Startup]

Some `progress.ini` file settings you can localize appear in the [Startup] section. Figure 10–1 shows the [Startup] section of the `progress.ini` file that Progress provides in the `DLC/bin` directory.

```
[Startup]
V6Display=no
;ImmediateDisplay=yes
;MultitaskingInterval=100
DefaultFont=MS Sans Serif, size=8
DefaultFixedFont=Courier New, size=8
```

Figure 10–1: [Startup] Section Of the progress.ini File

Table 10–1 describes the [Startup] section settings you might localize.

Table 10–1: Localizable Settings In the [Startup] Section Of the progress.ini File

Setting	Description
DefaultFont	The name of the default display font
DefaultFixedFont	The name of the default fixed-width display font
PrinterFont PrinterFont1 PrinterFont2 PrinterFont3	The name of the font the printer uses when you invoke the Progress 4GL OUTPUT TO statement's PRINTER option
PROMSGS	The name of the `promsgs` file the application uses. For example, to use the Swedish PROMSGS file, set PROMSGS to `DLC\prolang\swe\promsgs.swe`
PROSTARTUP	The name and path of the startup parameter (`.pf`) file Progress expects

10.1.2 [fonts]

Other `progress.ini` file settings you might localize appear in the [fonts] section. Figure 10–2 shows the [fonts] section of the `progress.ini` file Progress provides in the `DLC/bin` directory.

```
[fonts]
font0=Courier New, size=8
font1=MS Sans Serif, size=8
font2=Courier New, size=8
font3=Courier New, size=8
font4=MS Sans Serif, size=8
font5=MS Sans Serif, size=10
font6=MS Sans Serif, size=8, bold
font7=MS Sans Serif, size=8
```

Figure 10–2: [fonts] Section Of the progress.ini File

10.1.3 Specifying Scripts

For each font you specify in the `progress.ini` file's [Startup] section or [fonts] section, or in the Windows registry, you can specify a script, which corresponds to a code page. The syntax for specifying a script is:

SYNTAX

```
script=script-name
```

script-name

The name of the script. Use one of the following values: ansi, default, symbol, shiftjis, hangeul, gb2312, chinesebig5, oem, johab, hebrew, arabic, greek, turkish, vietnamese, thai, easteurope, russian, or baltic.

NOTE: Do not enclose the value in quotes.

An example of a font specification that specifies a script is:

```
font0=Courier New, size=8, script=russian
```

For more information on specifying scripts, see your Windows documentation.

10.2 The PROTERMCAP File

The PROTERMCAP file, used by UNIX character-interface applications only, tells Progress how to interact with a particular terminal, console, or terminal emulator. For more information on the PROTERMCAP file, see the *Progress Client Deployment Guide*.

10.3 The PROMSGS File

The PROMSGS file contains Progress run-time messages in a particular language. Each PROMSGS file has a file extension named after the language and resides in a directory named after the language. For example, the Hungarian PROMSGS file, `promsgs.hun`, resides in the `DLC/prolang/hun` directory, and the Japanese PROMSGS file, `promsgs.jpn`, resides in the `DLC/prolang/jpn` directory.

Progress supplies a PROMSGS file for each language it supports. When you package an international application, be sure to supply the PROMSGS files your customers need. When you configure an international application, you can tell Progress which PROMSGS file to use by using one of the following techniques:

- By setting a value for PROMSGS in the [Startup] section of the `progress.ini` file
- By setting the PROMSGS environment variable

For more information on setting the PROMSGS environment variable, see Chapter 7, “Using SQL-92.”

10.4 Parameter Files

Progress parameter (`.pf`) files, which contain lists of startup parameters, are particularly useful for deploying applications across multiple locales. You can supply your users with a parameter file for each locale. Progress itself supplies such a set of parameter files. These reside in the `DLC/prolang` directory in subdirectories for each language. For example, the Russian parameter file, `russian.pf`, resides in the `DLC/prolang/rus` directory.

Here is the Russian parameter file:

```
-d dmy
-lng "Russian"
-cpcase Basic
-cpcoll Russian

# -cpinternal (or -charset) - Specifies the code page for all internal
#                             data processing.
# MS-Windows Code Page
  -cpinternal 1251
# DOS Code Page
# -cpinternal ibm866
# ISO Standard Code Page
# -cpinternal iso8859-5
# UNIX Code Page
# -cpinternal koi8-r

# -cpstream (or -stream) - Specifies the code page for all stream files
# MS-Windows Code Page
  -cpstream 1251
# DOS Code Page
# -cpstream ibm866
# ISO Standard Code Page
# -cpstream iso8859-5
# UNIX Code Page
# -cpstream koi8-r
```

NOTE: In parameter files, Progress ignores lines that start with the pound sign (#).

The startup parameters you might localize appear in Figure 10–2.

Table 10–2: Localizable Startup Parameters

Startup Parameter	Description
Case Table (-cpcase)	The name of the case table used by the Progress database server and the Progress client
Collation Table (-cpcoll)	The name of the collation table used by the Progress database server and the Progress client
Date Format (-d)	The format in which Progress displays dates in the application
Fractional Separator (-numdec)	The character that separates the integer portion and the fractional portion of a decimal number.
Internal Code Page (-cpinternal)	The name of the code page used internally by the Progress database server and the Progress client
Language (-lng)	The value that the CURRENT–LANGUAGE function returns. For Progress applications translated using the Progress Translation Manager, the particular language you want the application to use.
Stream Code Page (-cpstream)	The name of code page Progress uses for stream I/O
Thousands Separator (-numsep)	The character that separates each group of three digits in the integer portion of a number.

You can also use parameter files with Progress utilities, such as PROSHUT and PROUTIL. For more information on these Progress utilities, see the *Progress Database Administration Guide and Reference*. For more information on startup parameters, see the *Progress Startup Command and Parameter Reference*.

10.5 The conmgr.properties File

The `conmgr.properties` file, another file whose values you might localize, stores properties of databases, configurations, and servergroups. These properties are described in Table 10–3.

Table 10–3: Localizable Properties In the conmgr.properties File

Property	Corresponding Startup Parameter	Description	Default Value
casetablename	-cpcase	The name of the case table used by the Progress database server and the Progress client	BASIC
collationtable	-cpcoll	The name of the collation table used by the Progress database	BASIC
conversionmap	-convmap	The name and path of the compiled CONVMAP file used by the Progress database server and the Progress client	DLC/convmap.cp
internalcodepage	-cpinternal	The name of the code page used internally by the Progress database server	ISO8859-1
logcharacterset	-cplog	The name of the code page used by the Progress database's log file	ISO8859-1

NOTE: Do not edit the `conmgr.properties` file directly. Instead, use the Progress Explorer. For more information on the Progress Explorer, see its online help.

For more information on the `conmgr.properties` file, see the *Progress Database Administration Guide and Reference*.

10.6 Report Builder Parameters

The Progress Report Builder provides parameters that let you localize its output. Table 10–4 describes these parameters.

Table 10–4: Localizable Report Builder Parameters

Parameter	Description
Fractional Separator (-numdec)	The character that separates the integer portion and the fractional portion of a decimal number.
Thousands Separator (-numsep)	The character that separates each group of three digits the integer portion of a number.

For more information on the Progress Report Builder, see the *Progress Results User's Guide for Windows*.

10.7 Additional Guidelines

When deploying and configuring an application across multiple locales, the guidelines in this section might save you time and trouble.

10.7.1 Running Applications Translated Using the Translation Manager

When running an application translated using the Progress Translation Manager, the application must connect to the translation databases in addition to the other databases. For more information, see the *Progress Translation Manager Guide*.

10.7.2 Using the UNIX Character Client With Double-byte Code Pages

When you use the UNIX character client with double-byte code pages, UNIX strips input characters to seven bits, which garbles double-byte data. To avoid this, before starting Progress, enter the following command at the command prompt:

```
stty -istrip
```

10.7.3 Specifying -cpinternal and -cpstream With Database Utilities

When you install Progress and specify default values for -cpinternal and -cpstream, the installation program writes these values to `DLC/startup.pf`, the main parameter file. You you subsequently run Progress or a database utility without specifying a value for -cpinternal or -cpstream, Progress uses the value in the main parameter file.

A

Progress Resources

This appendix describes the resources Progress provides for internationalization and localization. It contains the following sections:

- Progress Product Support For Multi-byte Characters
- Files In the DLC/prolang Directory
- Startup Parameters and Settings
- Progress 4GL
- Progress SQL-92
- Utilities
- Determining the Code Page

A.1 Progress Product Support For Multi-byte Characters

Table A–1 describes the support each Progress product provides for multi-byte characters.

Table A–1: Support For Multi-byte Characters *(1 of 3)*

Product	Multi-byte Support
AppBuilder	The AppBuilder—except for the Character Run window—supports double-byte characters. Widgets created with the AppBuilder can have double-byte characters in labels and in text. The Character Run window does not support double-byte or triple-byte characters.
AppServer	The AppServer supports double-byte and triple-byte characters.
Application Compiler	The Application Compiler supports double-byte and triple-byte characters. That is, the 4GL source code that the Application Compiler compiles can contain double-byte and triple-byte characters, and the resulting r-code supports double-byte and triple-byte characters.
Clients (graphical, character, and batch)	The graphical client and the UNIX character client support double-byte characters but not triple-byte characters. The Windows character client does not support multi-byte characters. The batch client supports UTF–8.
Data Dictionary	The Progress Data Dictionary supports multi-byte characters.
DataServers	The DataServer for ORACLE supports double-byte characters if the ORACLE DBMS is set up for Native Language Support (NLS). For more information on NLS, see the ORACLE documentation. The DataServer for ORACLE does not support UTF–8.

Table A–1: Support For Multi-byte Characters *(2 of 3)*

Product	Multi-byte Support
ESQL/C (SQL-89)	An ESQL/C program can have double-byte characters if the program's main procedure defines an external integer variable with the name "CharSet," and CharSet is assigned the name of the appropriate double-byte code page. Valid names, which reside in the file `sqlhli.h`, which PSC provides as part of the ESQL/C source, are: Japanese (128): SHIFTJIS_CHARSET or EUCJIS_CHARSET Korean (129): HANGEUL_CHARSET Simplified Chinese (134): GB2312_CHARSET Traditional Chinese (136): CHINESEBIG5_CHARSET Since Progress does not validate strings from ESQL/C programs, you must ensure that double-byte and triple-byte characters are not split.
HLC Applications	An HLC program can have double-byte and triple-byte characters. Since Progress does not validate strings from HLC programs, you must ensure that multi-byte characters are formed correctly and that they are not split. For example, Progress searches each character string for the NULL terminator. For each multi-byte character in the string, Progress searches only the lead byte. If the string ends in a multi-byte character and the NULL terminator resides (mistakenly) in a byte other than the position for a lead byte, Progress misses it and does not detect the end of the string correctly.
Procedure Editor	The Procedure Editor supports double-byte characters. To write a multi-byte Progress 4GL application, you can use any text editor that supports multi-byte characters. That is, you can use a multi-byte Unicode editor if you prefer.
Report Builder	The Report Builder does not support double-byte or triple-byte characters.
Roundtable	Roundtable supports double-byte characters.

Table A–1: Support For Multi-byte Characters *(3 of 3)*

Product	Multi-byte Support
SQL-89 Preprocessor	The SQL-89 Preprocessor supports multi-byte characters. Use the Language (-lng) startup parameter to specify the multi-byte code page for your program. Specify the -LANGUAGE option and the appropriate code-page name in the Preprocessor startup command. You can abbreviate the -LANGUAGE option as -L. It accepts these values: CP850, BIG5, GB2312, KSC5601, and SHIFTJIS. Note that the code-page names have no dashes (BIG5 and SHIFTJIS, not BIG-5 and SHIFT-JIS) when you use them in the Preprocessor startup command. For example, the following command starts the Preprocessor and specifies BIG–5, a code page for Traditional Chinese: `sqlcpp sqldemo.cc -L BIG5` If you do not specify the -LANGUAGE option, Progress uses the single-byte code page IBM850 as the default. This parameter setting is appropriate for all single-byte languages. Your C compiler must use the same code page as the one you specify with the -LANGUAGE option.
SQL-92	SQL-92 supports double-byte and triple-byte characters.
Translation Management System	The Translation Management System can manage translations to and from languages that use double-byte code pages.

A.2 Files In the DLC/prolang Directory

The DLC/prolang directory contains files that support internationalization and localization of applications. These files reside in subdirectories named for a particular language or locale. For example, DLC/prolang/rus contains files for Russian and DLC/prolang/tai contains files for Thai. For more information on the subdirectories and their contents, see the DLC/prolang/readme file.

Table A–2 describes the files in the DLC/prolang directory. In the table, *lng* represents a three-letter abbreviation for a particular locale.

Table A–2: Files In DLC/prolang

File	Description
lng/_tran.df *lng*/code-page.df	Defines the collation order for the respective language. For languages with multi-byte character sets, the collation order sorts all single-byte characters first. For example, the collation order for Japanese sorts English letters, then half-width Katakana characters, followed by multi-byte characters.
lng/empty.db	An empty database labelled with the appropriate code-page and collation table for the language. Copy this database to create your own database.
lng/language.pf	Contains typical startup parameters for a particular locale. That is, it sets the collation table and case table commonly used in a locale. Some *lng* subdirectories contain multiple parameter files become some locales that share a language use different code-pages, collation orders, or date formats.
lng/promsgs.lng	Contains Progress error messages for a particular locale. Set the PROMSGS environment variable to the appropriate file. For example, PROMSGS=%DLC%\prolang\sch\promsgs.sch causes an application to display Progress error messages in Simplified Chinese.

A.3 Startup Parameters and Settings

Table A–3 lists startup parameters useful when deploying applications across multiple locales. For more information on these startup parameters see the *Progress Startup Command and Parameter Reference*.

Table A–3: Startup Parameters *(1 of 4)*

Parameter	Description
Check Double-byte Enabled (-checkdbe)	Causes Progress to report a compile-time error whenever it finds a LENGTH function, SUBSTRING function, or OVERLAY statement that does not use the CHARACTER, RAW, COLUMN or FIXED option. Use -checkdbe when you modify a Progress application to support double-byte or Unicode characters.
Conversion Map (-convmap)	Specifies the conversion map file to use for code-page conversions, collation orders, and case conversions. The default conversion map file is `DLC/convmap.cp`.
Case Code Page (-cpcase)	Specifies a case table in the `convmap.cp` file to use for case conversions. The default is BASIC. Progress performs case conversions when you use the CAPS and LC functions, and when you use "!" in a character field format string, which tells Progress to convert all characters in the string to uppercase during input.
R-code Out Code Page (-cprcodeout)	Specifies the code page Progress uses when it writes r-code text segments. The default is the code page specified by -cpinternal.

Table A–3: Startup Parameters *(2 of 4)*

Parameter	Description
Collation Code Page (-cpcoll)	Specifies a collation table in the `convmap.cp` file to use for sorting. The default is BASIC.
Internal Code Page (-cpinternal)	Specifies the code page that Progress uses in memory and for graphical clients. The default is the ISO8859–1 code page, except when using Progress/400 Native 4GL, in which case the default is the ibm037 code page.
Log File Code Page (-cplog)	Specifies the code page that Progress uses when it writes to a log file. The default is the code page specified by -cpstream. When using Progress/400 Native 4GL, the default is the ibm037 code page.
Print Code Page (-cpprint)	Specifies the code page that Progress uses when it prints to a printer. The default is the code page specified by -cpstream. When using Progress/400 Native 4GL, the default is the ibm037 code page.
R-code In Code Page (-cprcodein)	Specifies the code page Progress uses when it reads r-code text segments. The default is the code page described in the r-code.
Stream Code Page (-cpstream)	Specifies the code page for stream I/O. The default code page is IBM850. When using Progress/400 Native 4GL, the default code page is ibm037.

Table A–3: Startup Parameters *(3 of 4)*

Parameter	Description
Terminal Code Page (-cpterm)	Specifies the code page for output to a character terminal. The default is the code page specified by -cpstream. When using Progress/400 Native 4GL, the default is the ibm037 code page.
Date Format (-d)	Specifies the date format—that is, the order in which the day, month, and year appear. Specify the format as a three-character string, composed of the letters d (for day), m (for month), and y (for year), in the preferred order. The default is “mdy”. The -d startup parameter corresponds to the DATE–FORMAT attribute of the SESSION system handle.
European Numeric Format (-E)	Specifies that the decimal separator is a comma and that the thousands separator is a period, for numeric values, for input and output. By default, the decimal separator is a period and the thousands separator is a comma. The -E startup parameter corresponds to the NUMERIC–FORMAT attribute of the SESSION system handle.
Language (-lng)	Specifies the initial value for the CURRENT–LANGUAGE function, which determines the r-code segment from which Progress reads character-string constants. Specify the language as a quoted character string. Use the quoted character strings defined when Translation Manager was run.
Fractional Separator (-numsep)	Specifies the character that separates the integer portion and the fractional portion of a decimal number.

Table A–3: Startup Parameters *(4 of 4)*

Parameter	Description
Thousands Separator (-numsep)	Specifies the character that separates each group of three digits in a number.
Century (-yy)	Specifies the starting year for the Progress two-digit year range of 100 years. The -yy startup parameter corresponds to the YEAR-OFFSET attribute of the SESSION system handle.

Table A–4 lists the Progress Explorer properties you can localize. For each property, the table gives the corresponding startup parameter.

Table A–4: Localizable Progress Explorer Properties

Progress Explorer Setting	Progress Explorer Default	Command Line Startup Parameter	Command Line Default
Code Page (internal)	ISO8859–1	-cpinternal	ISO8859-1
Not Applicable	IBM850	-cpstream	IBM850
Log Character Set	ISO8859–1	-cplog	IBM850
Case Table	BASIC	-cpcase	BASIC
Collation Table	BASIC	-cpcoll	BASIC
ConversionMap	`DLC/convmap.cp`	-convmap	`DLC/convmap.cp`

A.4 Progress 4GL

Table A–5 describes the Progress 4GL elements particularly useful for internationalizing and localizing applications.

Table A–5: 4GL Elements That Support Internationalization and Localization *(1 of 3)*

4GL Element	Description
ASC Function	Returns the value that a character has in the code page that Progress uses (default or the code page that -cpinternal specified). You can also specify a target and source code page, so that ASC converts the value a character has in the source code page to the value it has in the target code page before returning it.
CAPS Function	Changes any lowercase letters in a character string to uppercase. To do this, Progress uses a case table. Any double-byte characters in the string are not changed.
CHR Function	Returns the character that an integer represents in the code page that Progress uses (default or the code page that -cpinternal specified). You can also specify a target and source code page, so that CHR compares the value in the source code page and returns the character that the value represents in the target code page. **NOTE:** Do not use the CHR function for sorting. The code-page value of a character is not the same as its sort weight. The sort weight of a character resides in a collation table for the code page.
CODEPAGE–CONVERT Function	Converts a string value from one code page into another.
COLLATE Option of the FOR Statement, the OPEN QUERY Statement, and the PRESELECT Phrase	Computes the collation value of a string using one of several methods ("RAW," "CASE-SENSITIVE," "CASE-INSENSITIVE," or "CAPS") and the collation table you specify.
COMPARE Function	Compares two strings, using one of several methods ("RAW," "CASE-SENSITIVE," "CASE-INSENSITIVE," or "CAPS") and the collation table you specify.

Table A–5: 4GL Elements That Support Internationalization and Localization *(2 of 3)*

4GL Element	Description
COMPILE Statement LANGUAGES Option	Specifies for which languages to generate text segments in r-code. At compilation, Progress reads translated strings from the translation databases that you provide. The r-code has a text segment for each language.
CURRENT–LANGUAGE Function	Returns the value of the CURRENT–LANGUAGE variable.
CURRENT–LANGUAGE Statement	Sets the CURRENT–LANGUAGE variable with a string expression for the current Progress session.
DBCODEPAGE Function	Returns the code page that a connected database uses.
DBCOLLATION Function	Returns the name of the collation table that a connected database uses.
ENCODE Function	Encodes a character string. The string can contain multi-byte characters.
GET–CODEPAGES Function	Returns the list of code pages in the `convmap.cp` file that the current Progress session uses.
GET–COLLATIONS Function	Returns the list of collation tables in the `convmap.cp` file that the current Progress session uses.
LC Function	Changes any uppercase letters in a character string to lowercase. Progress uses a case table for this operation.
NUMERIC–DECIMAL–POINT Attribute of the SESSION System Handle	The character that separates the integer portion and the fractional portion of a decimal value.
NUMERIC–FORMAT Attribute of the SESSION System Handle	AMERICAN, EUROPEAN, or a character string consisting of the thousands separator followed by the fractional separator.
NUMERIC–SEPARATOR Attribute of the SESSION System Handle	The character that separates each group of three digits in a number.

Table A–5: 4GL Elements That Support Internationalization and Localization *(3 of 3)*

4GL Element	Description
RCODE–INFO Handle	The RCODE–INFO handle has two useful attributes for international applications: CODEPAGE, which returns the code page with which the r-code is labelled (that is, the -cpinternal value when the r-code was compiled), and LANGUAGES, which returns the languages for which there are text segments.
SESSION Handle	The SESSION handle has many attributes that indicate code-page setting and the numeric and date formats that the current Progress session uses.
SET–NUMERIC–FORMAT Method of the SESSION System Handle	Sets the NUMERIC–SEPARATOR and NUMERIC–DECIMAL attributes to arbitrary values simultaneously.

Table A–6 describes the 4GL elements that support multi-byte characters.

Table A–6: 4GL Elements That Support Multi-byte Characters *(1 of 4)*

4GL Element	Description
APPLY Statement	Performs the function of a specified key. The expression you APPLY can contain multi-byte characters.
ASC Function	For a double-byte character, returns a value greater than 255 and less than 65536. For a multi-byte character, returns a value up to 1571291 (HEX: 0xEFBFBF).
BEGINS Operator	Tests a character expression to see whether it begins with a second character expression. Both expressions can contain multi-byte characters.
CAPS Function	Changes any lowercase letters in a character string to uppercase. Any double-byte characters in the string are not changed.
CHR Function	For a value greater than 255 and less than 65536, checks for a lead-byte value. If the lead byte is valid, Progress creates and returns a double-byte character. Returns a character if the integer value you specify is within the range of 0 to 15712191. The function returns a null string if the integer value is outside this range, or if the value does not correspond to valid lead bytes or trail bytes.
ENCODE Function	Encodes a source character string and returns the encoded character string result. The source string can contain multi-byte characters. **NOTE:** If the ENCODE function processes a string under one code page and then processes the same string under a different code page, the results might not match. For more information, see the *Progress Language Reference*.
ENTRY Function	Returns a character string entry from a character-delimited list based on an integer position. The string can contain multi-byte characters since ENTRY determines the location based on a unit of measure in characters.

Table A–6: 4GL Elements That Support Multi-byte Characters *(2 of 4)*

4GL Element	Description
ENTRY Statement	Sets an element in a character-delimited list of strings to a value. All strings and character separators can contain multi-byte characters.
FILL Function	Generates a character string made up of a character string repeated a number of times. Both strings can contain multi-byte characters.
INDEX Function	Returns an integer that indicates the character position of the target string within the source string. The strings can contain multi-byte characters.
IS–LEAD–BYTE Function	Returns TRUE if the first byte of a string is valid as a lead byte of a multi-byte character. It returns FALSE if the first byte of a string is not valid as a lead byte.
LASTKEY Function	Returns the integer code of the most recent key sequence returned from the keyboard buffer. A key sequence is the set of keystrokes necessary to send one character or function key to the application. The LASTKEY values are available only after the Input Method Editor places the data in the keyboard buffer.
LC Function	Changes any uppercase letters in a character string to lowercase. Any double-byte characters in the string are not changed.
LEFT–TRIM Function	Removes specified leading characters in a character string. The string and trimmed characters can contain multi-byte characters.
LENGTH Function	Returns the number of characters or bytes in a string. The *type* option selects the unit for length. You can choose CHARACTER, COLUMN, or RAW (for bytes).
LOOKUP Function	Returns an integer that indicates the position of an expression in a list. It returns a 0 if the expression is not in the list. The list, expression, and delimiter can contain multi-byte characters.

Table A–6: 4GL Elements That Support Multi-byte Characters *(3 of 4)*

4GL Element	Description
MATCHES Operator	Compares a character expression to a pattern and returns a TRUE value if the expression satisfies the pattern criteria. The expression and pattern can contain multi-byte characters.
NUM–ENTRIES Function	Returns the number of items in a character-delimited list of strings. All strings can contain multi-byte characters.
OVERLAY Statement	Overlays a character expression in a field or variable starting at a given position, and optionally for a given length. The *type* option selects the unit for length and position. You can choose CHARACTER, RAW (for bytes), or COLUMN.
R–INDEX Function	Returns an integer that indicates the position of the target string within the source string. The search is performed right to left. Both strings can contain multi-byte characters.
READKEY Statement	Reads one key sequence[1] from an input source and sets the value of READKEY to the keycode of that key sequence. READKEY and LASTKEY values are available only after the Input Method Editor places the data in the keyboard buffer.
REPLACE Function	Lets you replace a substring with another substring. Both substrings can contain multi-byte characters.
RIGHT–TRIM Function	Removes specified trailing characters from a character string. The string and trimmed characters can contain multi-byte characters.
SEARCH Function	Searches the directories and libraries defined in the PROPATH environment variable for a file. The *opsys-file* can contain multi-byte characters.
STRING Function	Converts a value of any data type in *source* into a character value. The *source* can contain multi-byte characters.
SUBSTITUTE Function	Returns a character string that is made up of a base string plus substitutes arguments in the string. The base string and arguments can contain multi-byte characters.

Table A–6: 4GL Elements That Support Multi-byte Characters *(4 of 4)*

4GL Element	Description
SUBSTRING Function	Extracts a position of a character string from a field or variable. The *type* option selects the units for length and position. You can choose CHARACTER, RAW (for bytes), COLUMN, or FIXED. The FIXED type specifies that the maximum length is measured in bytes and the position in characters. If the last byte or bytes represent part of, but not all of, a multi-byte character, these bytes are excluded.
SUBSTRING Statement	Replaces characters in a field or variable with an expression you specify. You can replace the lead bytes or trail byte of the source string. The *type* option selects the unit for length and position. You can choose CHARACTER, FIXED, COLUMN, or RAW (for bytes). With the FIXED option, if the last byte or bytes represent part of, but not all of, a multi-byte character, these bytes are excluded.
TRIM Function	Removes leading and trailing single-byte spaces in a character string. The string and trimmed characters can contain multi-byte characters.

[1] A key sequence is the set of keystrokes necessary to send one character or function key to the application.

A.5 Progress SQL-92

Table A–7 describes the Progress SQL-92 elements that support internationalization and localization.

Table A–7: SQL-92 Language Elements That Support Internationalization and Localization *(1 of 11)*

Syntax	Description	Comment
{ CHARACTER | CHAR } [(*length*)]	The CHARACTER data type represents a null-terminated character field of length *length*.	*length* represents the number of characters.
{ CHARACTER VARYING | CHAR VARYING | VARCHAR } [(*length*)]	The VARCHAR data type represents a variable-length character field whose maximum length is *length*.	*length* represents the maximum number of characters.
= | <> | != | ^= | < | <= | > | >=	Relational operators specify how SQL compares expressions in basic and quantified predicates	To sort and compare CHARACTER data, the SQL server uses the numeric values in the collation table, not the numeric values in the code page.

Table A–7: SQL-92 Language Elements That Support Internationalization and Localization *(2 of 11)*

Syntax	Description	Comment
column_name [NOT] LIKE *string_constant* [ESCAPE *escape_character*]	The LIKE predicate searches for strings that have a certain pattern. The pattern is specified after the LIKE keyword in a string constant. The pattern can be specified by a string in which the underscore (_) and percent sign (%) characters have special semantics.	The LIKE predicate is multi-byte enabled. The *string_constant* and the *escape_character* may contain multi-byte characters, and the *escape_character* can be a multi-byte character. A percent sign (%) or an underscore (_) in the *string_constant* can represent a multi-byte character. However, the percent sign or underscore itself must be the single-byte ASCII encoding. The comparison is case-insensitive.
'*char-string*'	A character-string literal is a string of characters enclosed in single quotation marks (' '). To include a single quotation mark in a character-string literal, precede it with an additional single quotation mark.	A character string literal may contain multi-byte characters in the character set used by the SQL client. Only single-byte ASCII-encoded quote marks are valid in the syntax.

Table A–7: SQL-92 Language Elements That Support Internationalization and Localization *(3 of 11)*

Syntax	Description	Comment
{ d '*yyyy-mm-dd*' } | *mm-dd-yyyy* | *mm/dd/yyyy* | *mm-dd-yy* | *mm/dd/yy* | *yyyy-mm-dd* | *yyyy/mm/dd* | *dd-mon-yyyy* | *dd/mon/yyyy* | *dd-mon-yy* | *dd/mon/yy* { t '*hh:mi:ss*' } | *hh:mi:ss*[:*mls*]	SQL supports special formats for date and time literals. Basic predicates and the VALUES clause of INSERT statements can specify date literals directly for comparison and insertion into tables. In other cases, you need to convert date literals to the appropriate date-time data type with the CAST, CONVERT, or TO_DATE scalar functions	All text (names of days, months, ordinal number endings) in all types date-format literals must be in the English Language. The default date format is American. You can explicitly request another date format by using a format string.Time literals are in the English Language only.
ASCII (*char_expression*)	The scalar function ASCII returns the ASCII value of the first character of the given character expression.	The ASCII function depends on the code page and supports multi-byte characters. The function returns the character encoding integer value of the first character of *char_expression* in the current code page. Whether *char_expression* represents a literal string or a database field, the result depends on the code page of the database.

Table A–7: SQL-92 Language Elements That Support Internationalization and Localization *(4 of 11)*

Syntax	Description	Comment
CHAR (*integer_expression*)	The scalar function CHAR returns a character string with the first character having an ASCII value equal to the argument expression. CHAR is identical to CHR but provides ODBC-compatible syntax.	The CHAR function depends on the code page and supports single-byte and multi-byte characters. If *integer_expression* evaluates to an integer value that represents a character in the database code page, CHAR returns that character. Else, CHAR returns a NULL value.
CHR (*integer_expression*)	The scalar function CHR returns a character string with the first character having an ASCII value equal to the argument expression.	The CHR function depends on the code page and supports single-byte and multi-byte characters. If *integer_expression* evaluates to an integer value that represents a character in the database code page, CHR returns that character. Else, CHR returns a NULL value.
CONCAT (*char_expression* , *char_expression*)	The scalar function CONCAT returns a concatenated character string formed by concatenating argument one with argument two.	The two *char_expression* expressions and the result of the CONCAT function can contain multi-byte characters.

Table A–7: SQL-92 Language Elements That Support Internationalization and Localization *(5 of 11)*

Syntax	Description	Comment
CONVERT (*'data_type* [(*length*)] *'* , *expression*)	The Progress Extension scalar function CONVERT converts an expression to another data type. The first argument is the target data type. The second argument is the expression to be converted to that type.	When *data_type* is CHARACTER(*length*) or VARCHAR(*length*), the *length* specification represents the number of characters. The converted result can contain multi-byte characters.
GREATEST (*expression* , *expression* ...)	The scalar function GREATEST returns the greatest value among the values of the given expressions.	When the data type of an *expression* is either CHARACTER(*length*) or VARCHAR(*length*), the expression can contain multi-byte characters. The sort weight for each character is determined by the collation table in the database.
INITCAP (*char_expression*)	The scalar function INITCAP returns the result of the argument character expression after converting the first character to uppercase and the subsequent characters to lowercase.	A *char_expression* and the result can contain multi-byte characters. To convert the first character to uppercase and the subsequent characters to lowercase, Progress uses a case table in the `convmap.cp` file. The default case table is BASIC.

Table A–7: SQL-92 Language Elements That Support Internationalization and Localization *(6 of 11)*

Syntax	Description	Comment
INSERT (*string_exp1* , *start_pos* , *length* , *string_exp2*)	The scalar function INSERT returns a character string where *length* number of characters have been deleted from *string_exp1* beginning at *start_pos*, and *string_exp2* has been inserted into *string_exp1*, beginning at *start_pos*.	*string_exp1*, *string_exp2*, and the result might contain multi-byte characters, depending on the code page of the SQL server. The *length* argument specifies a character count.
INSTR (*char_expression1* , *char_expression2* [, *start_pos* [, *occurrence*]])	The INSTR (in string) scalar function searches character string *char_expression1* for the character string *char_expression2*. The search begins at *start_pos* of *char_expression1*. If *occurrence* is specified, then INSTR searches for the *n*th occurrence, where *n* is the value of the fourth argument.	A *char_expression* and the result can contain multi-byte characters.
LCASE (*char_expression*)	The scalar function LCASE returns the result of the argument character expression after converting all the characters to lowercase. LCASE is the same as LOWER but provides ODBC-compatible syntax.	A *char_expression* and the result may contain multi-byte characters. The conversion to lowercase conversion depends on the case table in the `convmap` file. The default case table is BASIC.

Table A–7: SQL-92 Language Elements That Support Internationalization and Localization *(7 of 11)*

Syntax	Description	Comment
LEAST (*expression* , *expression* , **...**)	The scalar function LEAST returns the lowest value among the values of the given expressions.	When the data type of an *expression* is either CHARACTER(*length*) or VARCHAR(*length*), the *expression* may contain multi-byte characters. The sort weight for each character depends on the collation table in the database.
LEFT (*string_exp* , *count*)	The scalar function LEFT returns the leftmost count of characters of *string_exp*.	The *string_exp* and the result can contain multi-byte characters. The function returns a character count.
LENGTH (*char_expression*)	The scalar function LENGTH returns the string length of the value of the given character expression.	*char_expression* may contain multi-byte characters. The function returns a character count.
LOCATE (*char_expr1* , *char_expr2* , **[** *start_pos* **]**)	The scalar function LOCATE returns the location of the first occurrence of *char_expr1* in *char_expr2*. If the function includes the optional integer argument *start_pos*, LOCATE begins searching *char_expr2* at that position. If the function omits the *start_pos* argument, LOCATE begins its search at the beginning of *char_expr2*.	*char_expr1* and *char_expr2* can contain multi-byte characters. The *start_pos* argument specifies a character position, not a byte position. Character comparisons use the collation table in the database.

Table A–7: SQL-92 Language Elements That Support Internationalization and Localization *(8 of 11)*

Syntax	Description	Comment
LPAD (*char_expression* , *length* [, *pad_expression*])	The scalar function LPAD pads the character string corresponding to the first argument on the left with the character string corresponding to the third argument. After the padding, the length of the result is *length*.	The *char_expression* and *pad_expression* can contain multi-byte characters. The *length* specifies a number of characters.
LTRIM (*char_expression* [, *char_set*])	The scalar function LTRIM removes all the leading characters in *char_expression* that are present in *char_set* and returns the resulting string. The first character in the result is guaranteed not to be in *char_set*. If you do not specify the *char_set* argument, leading blanks are removed.	The *char_expression*, the character set specified by *char_set*, and the result can contain multi-byte characters. Character comparisons are case-sensitive and depend on the collation table in the database.
PREFIX (*char_expression* , *start_pos* , *char_expression*)	The scalar function PREFIX returns the substring of a character string, starting from the position specified by *start_pos* and ending before the specified character.	Each *char_expression* and the result can contain multi-byte characters. The *start_pos* argument specifies a character position, not a byte position. Character comparisons are case-sensitive and depend on sort weights in the collation table in the database.
REPEAT (*string_exp* , *count*)	The scalar function REPEAT returns a character string composed of *string_exp* repeated *count* times.	*string_exp* and the result can contain multi-byte characters.

Table A–7: SQL-92 Language Elements That Support Internationalization and Localization *(9 of 11)*

Syntax	Description	Comment
REPLACE (*string_exp1* , *string_exp2* , *string_exp3*)	The scalar function REPLACE replaces all occurrences of *string_exp2* in *string_exp1* with *string_exp3*.	Each occurrence of *string_exp* and the result can contain multi-byte characters. Character comparisons are case-sensitive and depend on sort weights in the collation table in the database.
RIGHT (*string_exp* , *count*)	The scalar function RIGHT returns the rightmost count of characters of *string_exp*.	Each occurrence of *string_exp* and the result can contain multi-byte characters. Character comparisons are case-sensitive and depend on by sort weights in the collation table in the database.
RPAD (*char_expression* , *length* **[** , *pad_expression* **]**)	The scalar function RPAD pads the character string corresponding to the first argument on the right with the character string corresponding to the third argument. After the padding, the length of the result is equal to the value of the second argument *length*.	*char_expression* and *pad_expression* can contain multi-byte characters. *length* represents the number of characters in the result.

Table A–7: SQL-92 Language Elements That Support Internationalization and Localization *(10 of 11)*

Syntax	Description	Comment
RTRIM (*char_expression* [, *char_set*])	The scalar function RTRIM removes all the trailing characters in *char_expression* that are present in *char_set* and returns the resultant string. The last character in the result is guaranteed not to be in *char_set*. If you do not specify a *char_set*, trailing blanks are removed.	The *char_expression*, the character set specified by *char_set*, and the result can contain multi-byte characters. Character comparisons are case-sensitive and depend on the collation table in the database.
SUBSTR (*char_expression* , *start_pos* [, *length*])	The scalar function SUBSTR returns the substring of the character string corresponding to the first argument starting at *start_pos* and *length* characters long. If the third argument *length* is not specified, the substring starting at *start_pos* up to the end of *char_expression* is returned.	*char_expression* and the result can contain multi-byte characters. *length* specifies a number of characters. Character comparisons are case-sensitive depend on sort weights in the collation table in the database.
SUFFIX (*char_expression* , *start_pos* , *char_expression*)	The scalar function SUFFIX returns the substring of a character string starting after the position specified by *start_pos* and the second *char_expression*, to the end of the string.	Each *char_expression* and the result can contain multi-byte characters. The *start_pos* argument specifies a character position, not a byte position. Character comparisons are case-sensitive depend on sort weights in the collation table in the database.

Table A–7: SQL-92 Language Elements That Support Internationalization and Localization *(11 of 11)*

Syntax	Description	Comment
UCASE (*char_expression*)	The scalar function UCASE returns the result of the argument character expression after converting all the characters to uppercase. UCASE is identical to UPPER, but provides ODBC-compatible syntax.	A *char_expression* and the result can contain multi-byte characters. The conversion to uppercase depends on the case table in the convmap file. The default case table is BASIC.
UPPER (*char_expression*)	The scalar function UPPER returns the result of the argument character expression after converting all the characters to uppercase.	A *char_expression* and the result can contain multi-byte characters. The conversion to uppercase depends on the case table in the convmap file. The default case table is BASIC.

Table A–8 describes the format specifiers you can use with the SQL-92 TO_CHAR() and TO_DATE() functions.

Table A–8: Format Specifiers Allowed With TO_CHAR() and TO_DATE()
(1 of 4)

Format Specifier	Description	Comment
AM	Meridian indicator in the native language Without periods	English only
A.M.	Meridian indicator in the native language With periods	English only
CC	The century as a two-digit number Computed as one greater than the first two digits of the four-digit year	Considers 1900 to be in the 20th century, 2000 to be in the 21st century, etc.
SCC	The century as a two-digit number. Computed as one greater than the first two digits of the four-digit year BC dates are prefixed by "-"	Considers 1900 to be in the 20th century, 2000 to be in the 21st century, etc.
D	The day of the week as a one-digit number between 1 and 7	–
DAY	The day of the week in the native language Entire name in uppercase	English only
Day	The day of the week in the native language First letter only in uppercase	English only

Table A–8: Format Specifiers Allowed With TO_CHAR() and TO_DATE()
(2 of 4)

Format Specifier	Description	Comment
day	The day of the week in the native language First letter only in uppercase	English only
DD	The day of the month as a two-digit number between 01 and 31	–
DDD	The day of the year as a three-digit number between 001 and 365	–
DY	The day of the week as a three-character string in the native language	English only
HH	The hour of the day as a two-digit number between 01 and 12	–
HH12	The hour of the day as a two-digit number between 01 and 12	Synonym of HH
HH24	The hour of the day as a two-digit number between 00 and 23	–
J	The Julian day	–
MI	The minute of the hour as a two-digit number between 0 and 59	–
MM	The month as a two-digit number between 01 and 12	–
MON	The first three characters of the name of the month in the native language	English only

Table A–8: Format Specifiers Allowed With TO_CHAR() and TO_DATE()
(3 of 4)

Format Specifier	Description	Comment
MONTH	The first nine characters of the name of the month in the native language, right-padded with blanks	English only
PM	Meridian indicator in the native language Without periods	English only
P.M.	Meridian indicator in the native language With periods	English only
Q	The quarter of the year as a single digit between 1 and 4	–
SS	Seconds as a two-digit number between 00 and 59	–
SSSS	Seconds past midnight as a number between 0 and 86339	–
TH	Ordinal suffix appended to a number Replaced by ST, ND, RD or TH depending on the last digit of the number	English only
W	The week of the month as a single digit between 1 and 5	–
WW	The week of the year as a two-digit number between 01 and 52	–
Y	The year as a single digit	–
YY	The year as a two-digit number	–

Table A–8: Format Specifiers Allowed With TO_CHAR() and TO_DATE()
(4 of 4)

Format Specifier	Description	Comment
YYY	The year as a three-digit number	–
YYYY	The year as a four-digit number	–
Y,YYY	The year as a four-digit number A comma separates the first digit from the other digits	–

A.6 Utilities

Table A–9 lists the utilities mentioned in this book.

Table A–9: Utilities Used For Internationalization and Localization

Utility	Parameter or Qualifier
PROCOPY	None
PRODB	None
PROLIB	-list
PROUTIL	BULKLOAD CODEPAGE-COMPILER CONVCHAR CONV89 IDXBUILD IDXFIX WBREAK-COMPILER WORDRULES
SQLDUMP	None
SQLLOAD	None
SQLSCHEMA	None

For the complete syntax of each of these utilities except PROLIB, see the *Progress Database Administration Guide and Reference*. For the complete syntax of PROLIB, see the *Progress Client Deployment Guide*.

A.7 Determining the Code Page

When Progress encounters character data, it often must determine the code page the data is encoded in. Similarly, a developer working with databases, files, and other application components often has to determine the code page a component uses to code character data. Table A–10 lists typical application components and for each, describes how Progress determines the code page and how the developer can determine the code page.

Table A–10: Determining the Code Page *(1 of 4)*

Application Component	How Progress Determines the Code Page	How a Developer Can Determine the Code Page
Client or Server Memory[1]	Progress checks the value of the -cpinternal startup parameter. The default value is ISO8859–1. The code-page name must appear in the `convmap.cp` file.	A developer can check the setting of -cpinternal by using the SESSION:CPINTERNAL attribute.
Database	Progress reads the database's metaschema field, _Db._Db-xl-name. By default, the code page for all new databases is the code page of the empty database. There is an empty database for each locale.	A developer can connect to the database, then use the Data Administration utility, as described in Chapter 2, "Understanding Code Pages." A developer can change the name in this field with the PROUTIL CONVCHAR utility. For an empty database, a developer can load a `_tran.df` file that specifies a different name. A developer can use the DBCODEPAGE function, which displays the code page of the specified connected database.
Graphical Monitor and Keyboard	Progress checks the value of the -cpinternal startup parameter. The default value is ISO8859–1.	A developer can check the setting of -cpinternal by using the SESSION:CPINTERNAL attribute.

Table A–10: Determining the Code Page *(2 of 4)*

Application Component	How Progress Determines the Code Page	How a Developer Can Determine the Code Page
PROMSGS Files	Progress reads an internal label placed in each PROMSGS file.	A developer cannot change the value of this label, but can choose from a group of different PROMSGS files.
INPUT FROM..., OUTPUT TO..., INPUT–OUTPUT...	Progress checks the value of the -cpstream startup parameter. The default value is IBM850. The CONVERT and NO–CONVERT options override the current stream setting.	To determine the setting of -cpstream, a developer can use the SESSION:CPSTREAM attribute.
Character Terminals	Progress checks the value of the -cpterm parameter, and if no value is specified for this parameter, Progress checks the value of the -cpstream parameter. The default value for -cpstream is IBM850.	A developer can check the setting of -cpterm by using the SESSION:CPTERM attribute.
Procedure (`.p`) Files	Progress checks the value of the -cpstream parameter. The default value is IBM850.	A developer can check the setting of -cpstream by using the SESSION:CPSTREAM attribute.

Table A–10: Determining the Code Page *(3 of 4)*

Application Component	How Progress Determines the Code Page	How a Developer Can Determine the Code Page
R-code (`.r`) Files	Progress checks the code page value written in the r-code.	A developer can check the r-code setting of a particular file by using the RCODE–INFO:CODEPAGE attribute. The code page values specified with the R-code In Code Page (-cprcodein) and the R-code Out Code Page (-cprcodeout) startup parameters override the code page value written in the r-code. A developer can check the settings of -cprcodein and -cprcodeout by using the SESSION:CPRCODEIN and SESSION:CPRCODEOUT attributes, respectively. A developer can check the setting of -cpinternal by using the SESSION:CPINTERNAL attribute.

Table A–10: Determining the Code Page *(4 of 4)*

Application Component	How Progress Determines the Code Page	How a Developer Can Determine the Code Page
Table Dump (`.d`) and Data Definition (`.df`) files	Progress checks the value of the -cpstream parameter. The default value is IBM850. The Data Dictionary, however, implements its own rules when reading and writing these data and data definition files. When the Data Dictionary creates one of these files, it prompts you for a code page name. The Data Dictionary always appends a code-page name to a trailer in these files (either a name you specify or the name of the -cpstream code page). When the Data Dictionary reads one of these files, it checks for a code page in the trailer. If one is present, the Data Dictionary converts the data according to the name of the code page in the trailer. Otherwise, it uses the setting of -cpstream to determine the code-page name.	A developer can check the setting of -cpstream by using the SESSION:CPSTREAM attribute. The PROUTIL BULKLOAD utility also checks these files for a trailer. If there is a trailer, the PROUTIL BULKLOAD utility converts the data according to the name in the trailer. Otherwise, it uses the setting of -cpstream to determine the code-page name.

[1] This applies to the following executables: _progres, _prosrv, _mproshut, _proutil

B

Character Processing Table Formats

This appendix describes the format of the character processing tables. It contains the following sections:

- Character Attribute Table
- Case Table
- Collation Table
- Code-page Conversion Table
- Word-break Table

B.1 Character Attribute Table

Figure B–1 shows a typical character attribute table.

```
# This table contains the attributes for code page ibm850
CODEPAGE
CODEPAGE-NAME ibm850
TYPE "1"
ISALPHA
  /*000-015*/  000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
  /*016-031*/  000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
  /*032-047*/  000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
  /*048-063*/  000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
  /*064-079*/  000 001 001 001 001 001 001 001 001 001 001 001 001 001 001 001
  /*080-095*/  001 001 001 001 001 001 001 001 001 001 001 000 000 000 000 000
  /*096-111*/  000 001 001 001 001 001 001 001 001 001 001 001 001 001 001 001
  /*112-127*/  001 001 001 001 001 001 001 001 001 001 001 000 000 000 000 000
  /*128-143*/  001 001 001 001 001 001 001 001 001 001 001 001 001 001 001 001
  /*144-159*/  001 001 001 001 001 001 001 001 001 001 001 001 000 001 000 000
  /*160-175*/  001 001 001 001 001 001 000 000 000 000 000 000 000 000 000 000
  /*176-191*/  000 000 000 000 000 001 001 001 000 000 000 000 000 000 000 000
  /*192-207*/  000 000 000 000 000 000 001 001 000 000 000 000 000 000 000 000
  /*208-223*/  001 001 001 001 001 001 001 001 001 000 000 000 000 000 001 000
  /*224-239*/  001 001 001 001 001 001 000 001 001 001 001 001 001 001 000 000
  /*240-255*/  000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
ENDTABLE
ENDCODEPAGE
```

Figure B–1: Format Of the Character Attribute Table

The CODEPAGE keyword signals the start of the table entry. The CODEPAGE-NAME keyword precedes the name of the code page the character attribute table applies to. A code-page name cannot exceed 19 characters and can consist of the characters A–Z and a–z, the numerals 0–9, and the dash (-). The TYPE keyword tells Progress whether the code page is single byte or multi byte. For single byte, TYPE is 1. The ISALPHA keyword signals the start of a character attribute table. A value of 1 means the corresponding character is alphabetic; a value of 0 means the corresponding character is not alphabetic.

To build a character attribute table for a single-byte code page, provide 256 values in 16 rows of 16 values each. Be sure to format the rows of data exactly as shown.

The ENDTABLE keyword signals the end of the character attribute table. The ENDCODEPAGE keyword signals the end of the table entry.

B.2 Case Table

Figure B–2 shows the format of the case table.

```
#Optional comments starting with the pound sign
CASE
CODEPAGE-NAME codepage
CASETABLE-NAME casename
TYPE 1
UPPERCASE-MAP
  /*000-015*/  000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015
  /*016-031*/  016 017 018 019 020 021 022 023 024 025 026 027 028 029 030 031
  /*032-047*/  032 033 034 035 036 037 038 039 040 041 042 043 044 045 046 047
  /*048-063*/  048 049 050 051 052 053 054 055 056 057 058 059 060 061 062 063
  /*064-079*/  064 065 066 067 068 069 070 071 072 073 074 075 076 077 078 079
  /*080-095*/  080 081 082 083 084 085 086 087 088 089 090 091 092 093 094 095
  /*096-111*/  096 065 066 067 068 069 070 071 072 073 074 075 076 077 078 079
  /*112-127*/  080 081 082 083 084 085 086 087 088 089 090 123 124 125 126 127
  /*128-143*/  128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
  /*144-159*/  144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159
  /*160-175*/  160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175
  /*176-191*/  176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191
  /*192-207*/  192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207
  /*208-223*/  208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223
  /*224-239*/  192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207
  /*240-255*/  208 209 210 211 212 213 214 247 216 217 218 219 220 221 222 089
ENDTABLE
LOWERCASE-MAP
  /*000-015*/  000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015
                                           .
                                           .
                                           .
  /*240-255*/  240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255
ENDTABLE
ENDCASE
```

Figure B–2: Format Of the Case Table

The CASE keyword signals the start of the table entry. The CASE keyword takes the optional argument NONASCII, used for an EBCDIC code page or for any other code page that does not use standard case rules for the ASCII characters—that is, the characters with numeric values in the range 0–127. The CODEPAGE-NAME keyword precedes the name of the code page the case table applies to. The CASETABLE-NAME keyword precedes the name of the case table. This name cannot exceed 19 characters and can include the characters A–Z and a–z, the numerals 0–9, and the dash (-).

In the UPPERCASE-MAP section, each number represents the value of the corresponding character's uppercase equivalent in the case table's code page. Similarly, in the LOWERCASE-MAP section, each number represents the value of the corresponding character's lowercase equivalent in the case table's code page.

To build a case table for a single-byte code page, provide 256 values in 16 rows of 16 cells. Be sure to format the rows of data exactly as shown.

The ENDTABLE keyword signals the end of the character attribute table. The ENDCASE keyword signals the end of the table entry.

B.3 Collation Table

Figure B–3 shows the format of the collation table.

```
#Optional comments starting with pound sign
COLLATION
CODEPAGE-NAME codepage
COLLATION-NAME collation
COLLATION-TRANSLATION-VERSION 1.0-16
CASE-INSENSITIVE-SORT
  /*000-015*/  000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015
  /*016-031*/  016 017 018 019 020 021 022 023 024 025 026 027 028 029 030 031
  /*032-047*/  032 033 034 035 036 037 038 039 040 041 042 043 044 045 046 047
  /*048-063*/  048 049 050 051 052 053 054 055 056 057 058 059 060 061 062 063
  /*064-079*/  064 065 066 067 068 069 070 071 072 073 074 075 076 077 078 079
  /*080-095*/  080 081 082 083 084 085 086 087 088 089 090 091 092 093 094 095
  /*096-111*/  096 065 066 067 068 069 070 071 072 073 074 075 076 077 078 079
  /*112-127*/  080 081 082 083 084 085 086 087 088 089 090 123 124 125 126 127
  /*128-143*/  176 177 178 179 180 185 186 187 188 191 192 193 194 195 196 197
  /*144-159*/  200 201 202 203 204 205 206 213 217 218 219 220 223 242 254 158
  /*160-175*/  255 173 189 156 207 190 221 245 249 184 229 174 199 240 199 238
  /*176-191*/  248 241 253 252 239 230 244 250 247 251 229 175 172 171 243 168
  /*192-207*/  065 065 065 065 065 143 146 067 069 069 069 069 073 073 073 073
  /*208-223*/  068 078 079 079 079 079 079 158 157 085 085 085 085 089 222 000
  /*224-239*/  065 065 065 065 065 143 146 067 069 069 069 069 073 073 073 073
  /*240-255*/  068 078 079 079 079 079 079 246 157 085 085 085 085 089 222 089
ENDTABLE
CASE-SENSITIVE-SORT
  /*000-015*/  000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015
                                          .
                                          .
                                          .
  /*240-255*/  208 110 111 111 111 111 111 246 155 117 117 117 117 121 231 121
ENDTABLE
ENDCOLLATION
```

Figure B–3: Format Of the Collation Table

The keyword COLLATION signals the start of a collation table entry. CODEPAGE-NAME precedes the name of the code page the collation table applies to. COLLATION-NAME precedes the name of the collation table. This name cannot exceed 19 characters and can include the characters A–Z and a–z, the numerals 0–9, and the dash (-).

COLLATION-TRANSLATION-VERSION precedes a value that Progress uses internally. Progress Software Corporation recommends that for single-byte collation tables you specify the value “1.1-16” (without the quotes).

The CASE-INSENSITIVE-SORT and CASE-SENSITIVE-SORT tables are identical to those used for a database and operate similarly.

To build a collation table for a single-byte code page, provide 256 values in 16 rows of 16 values. Be sure to format the rows of data exactly as shown.

NOTE: For the German character sharp-s (written “ß” or “ss”), supply the value 0.

The ENDTABLE keyword signals the end of the collation table. The ENDCOLLATION keyword signals the end of the table entry.

B.4 Code-page Conversion Table

Figure B–4 shows the format of the code-page conversion table.

```
# Optional comments starting with the pound sign
CONVERT
SOURCE-NAME source-codepage
TARGET-NAME target-codepage
TYPE "1"
  /*000-015*/  000 001 002 003 055 045 046 047 022 005 037 011 012 013 014 015
  /*016-031*/  016 017 018 019 060 061 050 038 024 025 028 039 063 029 030 031
  /*032-047*/  064 090 127 123 091 108 080 125 077 093 092 078 107 096 075 097
  /*048-063*/  240 241 242 243 244 245 246 247 248 249 122 094 076 126 110 111
  /*064-079*/  124 193 194 195 196 197 198 199 200 201 209 210 211 212 213 214
  /*080-095*/  215 216 217 226 227 228 229 230 231 232 233 186 224 187 176 109
  /*096-111*/  121 129 130 131 132 133 134 135 136 137 145 146 147 148 149 150
  /*112-127*/  151 152 153 162 163 164 165 166 167 168 169 192 079 208 161 255
  /*128-143*/  104 220 081 066 067 068 071 072 082 083 084 087 086 088 099 103
  /*144-159*/  113 156 158 203 204 205 219 221 223 236 252 112 177 128 191 007
  /*160-175*/  069 085 206 222 073 105 154 155 171 175 095 184 183 170 138 139
  /*176-191*/  043 044 009 033 040 101 098 100 180 056 049 052 051 074 178 036
  /*192-207*/  034 023 041 006 032 042 070 102 026 053 008 057 054 048 058 159
  /*208-223*/  140 172 114 115 116 010 117 118 119 035 021 020 004 106 120 059
  /*224-239*/  238 089 235 237 207 239 160 142 174 254 251 253 141 173 188 190
  /*240-255*/  202 143 027 185 182 181 225 157 144 189 179 218 250 234 062 065
ENDTABLE
ENDCONVERT
```

Figure B–4: Format Of the Code-page Conversion Table

The CONVERT keyword signals the beginning of a code-page conversion table entry. The NOINVERSE option of the CONVERT keyword tells Progress not create the table for the inverse conversion. If NOINVERSE does not appear, Progress automatically creates the table for the inverse conversion. NOINVERSE is useful for one-way conversions such as character sets for terminals and printers. The SOURCE-NAME keyword precedes the name of the source code page. The TARGET-NAME keyword specifies the name of the target code page. The TYPE statement specifies a conversion algorithm. For a conversion between two single-byte code pages, set TYPE to 1.

To build a table to convert between two single-byte code pages, provide 256 values in 16 rows of 16 values. Be sure to format the rows of data exactly as shown.

The ENDTABLE keyword signals the end of the code-page conversion table. The ENDCONVERT keyword signals the end of the table entry.

To convert a character from one code page to another, Progress looks in the code-page conversion table for the cell corresponding to the character's numeric value in the source code page. The cell contains the numeric value of the character in the target code page. For example, consider converting the character "ä" from ISO8859–1 to IBM850. In ISO8859–1, "ä" has the value 228. Figure B–5 shows part of the ISO 8859–1 to IBM850 conversion table. To perform the conversion, Progress looks in the table for cell 228 and finds the value 132. This value represents the numeric value of "ä" in IBM850.

```
#CONVMAP.dat Version 1.01
# This contains the data needed to convert from
# iso8859-1 to ibm code page 850
CONVERT
SOURCE-NAME "iso8859-1"
TARGET-NAME "ibm850"
TYPE 1
  /*000-015*/  000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015
  /*016-031*/  016 017 018 019 020 021 022 023 024 025 026 027 028 029 030 031
                                        .
                                        .
                                        .
  /*224-239*/  133 160 131 198 132 134 145 135 138 130 136 137 141 161 140 139
  /*240-255*/  208 164 149 162 147 228 148 246 155 151 163 150 129 236 231 152
ENDTABLE
ENDCONVERT
```

Figure B–5: Portion Of a Code-page Conversion Table

B.5 Word-break Table

The syntax of the word-break table is:

SYNTAX

```
[ #define symbolic-name symbol-value ] . . .

[ Version = 9
   Codepage = codepage-name
   wordrules-name = wordrules-name
   type = table-type
]

word_attr =
{
  { char-literal | hex-value | decimal-value } , word-delimiter-attribute
      [ , { char-literal | hex-value | decimal-value }
          , word-delimiter-attribute ] . . .
};
```

symbolic-name

The name of a symbol.

For example: DOLLAR-SIGN

symbol-value

The value of the symbol.

For example: '$'

NOTE: Although some versions of Progress let you compile word-break tables that omit all items within the second pair of square brackets, Progress Software Corporation (PSC) recommends that you always include these items. If the source-code version of a compiled word-break table lacks these items, and the associated database is not so large as to make this unfeasible, PSC recommends that you add these items to the table, recompile the table, reassociate the table with the database, and rebuild the indexes.

codepage-name

The name, not surrounded by quotes, of the code page the word-break table is associated with. The maximum length is 20 characters.

For example: UTF-8

wordrules-name

The name, not surrounded by quotes, of the compiled word-break table. The maximum length is 20 characters.

For example: utf8sample

table-type

The number 3.

NOTE: Some versions of Progress allow a table type of 1 or 2. Although these are still supported, Progress Software Corporation (PSC) recommends, if feasible, that you change the table type to 3, recompile the word-break table, reassociate it with the database, and rebuild the indexes.

char-literal

A character within single quotes or a *symbolic-name*, which represents a character in the code page.

For example: '#'

hex-literal

A hexadecimal value or a *symbolic-name*, which represents a character in the code page.

For example:0xAC

decimal-literal

A decimal value or a *symbolic-name*, which represents a character in the code page.

For example: 39

word-delimiter-attribute

In what context the character is a word delimiter.

Table B–1 describes the word-delimiter attributes.

Table B–1: Word-delimiter Attributes *(1 of 2)*

Word Delimiter Attribute	Description	Default
LETTER	Always part of a word.	Assigned to all characters that the current attribute table defines as letters. In English, these are the uppercase characters A–Z and the lowercase characters a–z.
DIGIT	Always part of a word.	Assigned to the characters 0–9.
USE_IT	Always part of a word.	Assigned to the following characters: • Dollar sign ($) • Percent sign (%) • Number sign (#) • At symbol (@) • Underline (_)
BEFORE_LETTER	Part of a word only if followed by a character with the LETTER attribute. Else, treated as a word delimiter.	–
BEFORE_DIGIT	Treated as part of a word only if followed by a character with the DIGIT attribute.	Assigned to the following characters: • Period (.) • Comma (,) • Hyphen (-) For example, "12.34" is one word, but "ab.cd" is two words.

Table B–1: Word-delimiter Attributes *(2 of 2)*

Word Delimiter Attribute	Description	Default
BEFORE_LET_DIG	Treated as part of a word only if followed by a character with the LETTER or DIGIT attribute.	–
IGNORE	Ignored.	Assigned to the apostrophe (’). For example, "John’s" is equivalent to "Johns."
TERMINATOR	Word delimiter.	Assigned to all other characters.

Index

A

Accelerators
  with multi-byte labels 8–37, 8–38

ActiveX clients 9–3

Addresses
  international formats 4–13
  postal codes
    CHARACTER data type 4–13

AppBuilder 5–13
  multi-byte support 8–4, A–2
  multiple layouts 5–18

Application Compiler
  multi-byte support 8–4, A–2

APPLY statement 8–13
  multi-byte support A–13

AppServers
  multi-byte support 8–4

ASC function A–10
  multi-byte support 8–28, A–13

ASCII function 7–16, A–19

Audience 2–xiii

B

Batch clients
  multi-byte support 8–4, A–2

BEFORE_DIGIT
  word-delimiter attribute B–10

BEFORE_LET_DIG
  word-delimiter attribute B–11

BEFORE_LETTER
  word-delimiter attribute B–10

BEGINS operator
  multi-byte support A–13

Bold typeface
  as typographical convention 2–xv

BULKLOAD qualifier
  PROUTIL utility A–32

C

Calendars
  international conventions 4–11

CAPS function A–10
  multi-byte support A–13

Carons 6–5

Case Code Page (-cpcase) startup parameter 4–16, A–6

Case conversions 4–16

CASE statement 4–2

Case tables 3–6, 4–16
  format B–3

CASE-INSENSITIVE-SORT collation tables 3–18, 4–15

CASE-SENSITIVE-SORT collation tables 3–18, 4–15

casetablename property
  conmgr.properties file 10–8

Celsius (temperature scale) 4–9

Century (-yy) startup parameter A–9

CHAR function 7–17, A–20

Character attribute tables 3–5
  format B–2

Character clients
  multi-byte support 8–4, A–2

Character conversions
  Data Dictionary rules A–36
  Progress executables A–35

CHARACTER data type 7–14, A–17

Character processing
  byte level 4–4

Character processing tables
  definition of 3–1

Character sets
  definition of 1–4, 2–5

Character terminals
  code page of 2–14

Character-string literal 7–15, A–18

CHARSET attribute
  SESSION handle A–33

CharSet variable A–3

Check Double-byte Enabled (-checkdbe) startup parameter A–6

-checkdbe startup parameter A–6

chkdot.p utility 2–12

CHR function 7–17, 8–12, 8–14, A–10, A–20
  multi-byte support 8–28, A–13
  sorting 4–14

Clients
  multi-byte support 8–4, A–2

Code page
  undefined
    definition of 2–11

Code pages
  character terminals A–34
  client or server memory A–33
  COMPILE statement 4–18
  data definition (.df) files A–36
  databases A–33
  definition of 2–2
  graphical devices A–33
  INPUT/OUTPUT statements A–34
  procedure (.p) files A–34
  PROMSGS file A–34
  r-code (.r) files A–35
  table dump (.d) files A–36

Code points
  definition of 2–4

CODEPAGE attribute
  RCODE-INFO handle 4–19

Code-page conversion 9–2
  character conflicts 6–5
    scanning for 6–8
  databases 6–10
  diagram 2–7

Code-page conversion tables 3–11
format B–6

Code-page trailers 6–11

CODEPAGE-COMPILER qualifier
PROUTIL utility A–32

CODEPAGE-CONVERT function A–10

COLLATE option 9–5, 9–10, A–10

Collation Code Page (-cpcoll) startup parameter A–7

COLLATION keyword
collation tables 3–18

Collation tables 3–9, 4–15
CASE-INSENSITIVE-SORT 3–18
CASE-SENSITIVE-SORT 3–18
dumping 3–16
editing 3–18
format B–4
loading 3–18
modifying 3–15
_tran.df file A–5

Collations
indexes 4–16
international 4–14

collationtable property
conmgr.properties file 10–8

Colors
selecting 5–7

COMPARE function A–10

COMPILE statement
code pages 4–18
LANGUAGES option A–11

Compiling
translated applications 4–18

Composed characters
definition of 9–5, 9–10

CONCAT function 7–17, A–20

CONTAINS operator
WHERE option
record phrase 3–21

CONV89 qualifier
PROUTIL utility A–32

CONVCHAR qualifier
PROUTIL utility A–32

Conversion Map (-convmap) startup parameter A–6

conversionmap property
conmgr.properties file 10–8

Conversions
case 4–16

CONVERT function 7–18, A–21

-convmap startup parameter 7–2, A–6

convmap.dat file 3–2
compiling 3–14
modifying 3–13, 3–15
providing access to 3–14

-cpcase startup parameter 4–16, 7–2, 10–7, A–6

-cpcoll startup parameter 7–2, 10–7, A–7

CPINTERNAL attribute
SESSION handle A–33

-cpinternal startup parameter 6–13, 7–2, A–7

-cplog startup parameter 7–2, A–7

-cpprint startup parameter A–7

-cprcodein startup parameter A–7

-cprcodeout startup parameter A–6

CPSTREAM attribute

-cpstream startup parameter 6–13, 7–2, 10–7, A–7

CPTERM attribute
  SESSION handle A–34

-cpterm startup parameter A–8

Currencies
  data types 4–10
  international formats 5–8

CURRENT-LANGUAGE function 4–2, A–11

CURRENT-LANGUAGE statement A–11

D

-d startup parameter 4–11, A–8

Data definition (.df) files 2–9

Data Dictionary
  multi-byte support 8–4, A–2

Data types
  INTEGER 4–11

Database collation tables 4–15

Database names
  restrictions 6–2

Database servers 7–2

Databases 1–6
  collation 4–16
  empty 6–2, A–5

DataServers
  multi-byte support 8–4, A–2

Date data
  storing 4–11

DATE data type 4–11

Date Format (-d) startup parameter 4–11, 10–7, A–8

DATE function 4–11

Date-format literals 7–16, A–19

Dates
  handling international standards
    sample procedure 4–11
  international conventions 4–11
  processing 4–11

DAY function 4–11

DBCODEPAGE function A–11
  4GL 2–13

DBCOLLATION function A–11

DECIMAL data type
  display format 4–8

Decimal point
  numeric formats 4–8

DefaultFixedFont setting
  progress.ini file 10–3

DefaultFont setting
  progress.ini file 10–3

Diereses 6–5

DIGIT
  word-delimiter attribute B–10

Display formats 8–15

Double-byte characters
  collation order 8–34, 8–36, 8–37
  definition of 8–2
  sorting 8–34, 8–36

Double-byte code pages 2–4
  definition of 8–2

Double-byte enabled
  LENGTH function 8–24

E

-E startup parameter 4–8, A–8

8-bit characters
code page of 2–12

ENCODE function A–11
multi-byte support 8–29, A–13

ENTRY function
multi-byte support A–13

ENTRY statement
multi-byte support A–14
multi-byte support of A–14

Error messages
displaying descriptions 2–xxiii

ESQL/C
multi-byte support 8–5, A–3

Euro symbol 6–5

European Numeric Format (-E) startup parameter 4–8, A–8

Example procedures 2–xx

Explorer
Progress 7–2

F

Field labels
localized 5–8

FILL function
multi-byte support A–14

Fonts
setting in progress.ini 10–3

FOR statement
COLLATE option A–10

FORMAT option
displaying dates 4–11

FORMAT phrase 8–15
multi-byte support 8–29

Fractional separator
numeric formats 4–8

Fractional Separator (-numdec) startup parameter 10–7

G

Gaiji characters 8–33

GET-CODEPAGES function A–11

GET-COLLATIONS function A–11

Graphical clients
multi-byte support 8–4, A–2

GREATEST function 7–18, A–21

H

Help
Progress messages 2–xxiii

HLC applications
multi-byte support 8–5, A–3

I

IDXBUILD qualifier
PROUTIL utility A–32

IDXFIX qualifier
PROUTIL utility A–32

IGNORE
word-delimiter attribute B–11

Images
cross-cultural 5–11, 5–12

IME. See Input Method Editors

INCLUDE directives 3–2

INDEX function
multi-byte support 8–29, A–14

Indexes
rebuilding 3–19, 3–29

INI2REG utility 10–2

INITCAP function 7–18, A–21

Initialization (.ini) files 10–2

INPUT FROM statement
4GL 2–9

Input method editors 4–6, 5–10, 8–37
definition of 8–7

Input methods
definition of 8–7

INSERT function 7–19, A–22

Installation program
Progress 10–2

INSTR function 7–19, A–22

INTEGER data type 4–11

Internal Code Page (-cpinternal) startup parameter 10–7, A–7

internalcodepage property
conmgr.properties file 10–8

Internationalized applications
definition of 1–3

IS-LEAD-BYTE function 8–27
multi-byte support 8–29, A–14

Italic typeface
as typographical convention 2–xv

J

Java clients 9–3

K

Keyboards
character sets 5–3
mapping 4–6

Keystrokes 2–xv

L

Language (-lng) startup parameter 8–6, 10–7, A–8

Language issues
character sets 5–10
direction text is read 5–10
input method editors 5–10
translation 5–19
user interface text 5–10

LANGUAGE option A–4

LANGUAGES attribute
RCODE-INFO handle 4–19

LANGUAGES option
COMPILE statement A–11

LASTKEY function 8–8, 8–13
multi-byte support A–14

Layout. See Screen layout

LC function A–11
multi-byte support A–14

LCASE function 7–19, A–22

Lead bytes
definition of 8–3

LEAST function 7–20, A–23

LEFT function 7–20, A–23

LENGTH function A–14, A–23

LEFT-TRIM function
multi-byte support A–14

LENGTH function 7–20, 8–24
multi-byte support 8–29

LETTER
word-delimiter attribute B–10

Libraries
Progress
code page of 2–17

Ligatures 6–5

LIKE predicate 7–15, A–18

-lng startup parameter 8–6, A–8

Locale
definition of 1–4

Localization
graphics 5–11
multiple layouts 5–18
user interface text 5–4

Localized application
definition of 1–4

LOCATE function 7–20, A–23

Log File Code Page (-cplog) startup parameter A–7

logcharacterset property
conmgr.properties file 10–8

LOOKUP function
multi-byte support A–14

LPAD function 7–21, A–24

LTRIM function 7–21, A–24

M

Manual
syntax notation 2–xvi

Manual, organization of 2–xiii

MATCHES operator
multi-byte support A–15

Messages
displaying descriptions 2–xxiii

Metric system 4–9

Monitors
different models 5–2

Monospaced typeface
as typographical convention 2–xv

MONTH function 4–11

Multi-byte characters
definition of 8–2

Multi-byte code pages
definition of 8–2

Multilingual data 9–2, 9–3

Multiple layouts 5–18

N

N-byte characters
definition of 8–2

N-byte code pages
definition of 8–2

Negative numbers
international notation 4–10

Non-ASCII characters 6–2

NUM-ENTRIES function
multi-byte support A–15

Numeric formats
internal processing 4–8
international standards 4–8

NUMERIC-DECIMAL-POINT attribute
SESSION handle A–11

NUMERIC-FORMAT attribute
SESSION handle 4–8, 4–11, A–11

NUMERIC-SEPARATOR attribute
SESSION handle A–11

-numsep startup parameter A–8, A–9

O

OPEN QUERY statement
COLLATE option A–10

ORACLE DataServer
multi-byte support 8–4, A–2

OUTPUT TO statement
4GL 2–9

OVERLAY statement
multi-byte support 8–29, A–15

P

Paper sizes
international standards 4–7

Parameter files 1–7, A–5
international 10–6

Phone numbers
international conventions 5–9

Postal codes
CHARACTER data type 4–13

PREFIX function 7–21, A–24

PRESELECT phrase
COLLATE option A–10

Print Code Page (-cpprint) startup parameter A–7

PrinterFont setting
progress.ini file 4–7, 10–3

Printers
code page of 2–16

Printing
paper sizes 4–7
setting fonts in progress.ini 10–3

Procedure Editor
multi-byte support 8–5, A–3

Procedures
examples of xx

PROCOPY utility A–32

PRODB utility 9–8, A–32

Progress AppServers
multi-byte support A–2

Progress databases
code page of 2–13

Progress Explorer 7–2

progress.ini file 4–7

PROLIB utility 2–17, A–32

PROMSGS environment variable 10–5, A–5

PROMSGS file 1–7, 10–5
database server 7–3
localized editions 1–7

PROMSGS files A–5

PROMSGS setting
progress.ini file 10–3

PROSERVE 7–2

PROSTARTUP setting
progress.ini file 10–3

PROTERMCAP file 2–9, 10–5

PROUTIL utility A–32
  BULKLOAD qualifier 6–12, A–36
  CODEPAGE-COMPILER qualifier 3–14
  CONVCHAR CHARSCAN qualifier 2–10, 6–8
  CONVCHAR CONVERT qualifier 6–10
  CONVCHAR qualifier A–33
  IDXBUILD qualifier 3–19, 3–29, 6–3
  IDXFIX qualifier 3–29
  WBREAK-COMPILER qualifier 3–27
  WORD-RULES qualifier 3–28

p-snhndl1.p 4–11

R

Radix 4–8

R-code
  compiling 8–4, A–2

R-code In Code Page (-cprcodein) startup parameter A–7

R-code Out Code Page (-cprcodeout) startup parameter A–6

RCODE-INFO handle 4–18
  CODEPAGE attribute 4–19, A–12
  LANGUAGES attribute 4–19, A–12

READKEY statement 8–8, 8–13, 8–14
  multi-byte support A–15

Relational operators 7–14, A–17

REPEAT function 7–21, A–24

REPLACE function 7–22, A–25
  multi-byte support A–15

Report Builder
  localizable parameters 10–9
  multi-byte support 8–5, A–3

RIGHT function 7–22, A–25

RIGHT-TRIM function
  multi-byte support A–15

R-INDEX function
  multi-byte support A–15

Rounding methods
  cultural conventions 5–9

Roundtable
  multi-byte support A–3

RPAD function 7–22, A–25

RTRIM function 7–23, A–26

S

Screen layout 5–2, 5–4

Screens
  differences 5–2

Scripts
  specifying
    in the progress.ini file 10–4

SEARCH function
  multi-byte support A–15

Servers
  database 7–2

SESSION handle 4–18, A–12

SET statement 8–14

7-bit characters
  code page of 2–12

Single-byte characters
  definition of 8–2

Single-byte code pages 2–4
  definition of 8–2

SmartObjects
  instance attributes 5–15

Sockets 2–10

Sorting
  double-byte characters 8–34, 8–36
  international data 4–14

Sounds
selecting 5–7

Source code page
definition of 2–10

SQL-89 Preprocessor
multi-byte support 8–6, A–4

SQL-92
multi-byte support 8–6, A–4

Startup parameters
localizable 7–2

Storing date data 4–11

Stream Code Page (-cpstream) startup parameter A–7

Streams 2–9

STRING function
multi-byte support 8–29, A–15

stty command (UNIX)
-istrip flag 8–37

SUBSTITUTE function
multi-byte support A–15
text translation 5–21

SUBSTR function 7–23, A–26

SUBSTRING function
multi-byte support 8–30, A–16

SUBSTRING statement
multi-byte support 8–30, A–16

SUFFIX function 7–23, A–26

Syntax notation 2–xvi

T

Table dump (.d) files 2–9
code page of 2–17
loading 6–11

Target code pages
definition of 2–10

Terminal Code Page (-cpterm) startup parameter A–8

Terminals
character
code page of 2–14

TERMINATOR
word-delimiter attribute B–11

Text editors 8–5, A–3

Text files
code page of 2–17

Text formats
addresses 5–5
currency values 5–8
dates and times 5–6
symbols and abbreviations 5–10

Thousands Separator (-numsep) startup parameter 10–7, A–8, A–9

Time
CHARACTER data type 4–13
international standards 4–13

Time-format literals 7–16, A–19

TO_CHAR function
format specifiers A–28

TO_DATE function
format specifiers A–28

Trail bytes
definition of 8–3
sorting 8–35

Trailers
codepage 6–11

_tran.df file 3–18, A–5

Translation
labeling strings for 4–5
user interface text 5–19

Translation Management System 1–6, 4–5
multi-byte support 8–6, A–4

Translation Manager 1–6, 10–10

TRIM function
multi-byte support A–16

Triple-byte characters
definition of 8–2

Triple-byte code pages 2–4
definition of 8–2

Typographical conventions 2–xv

U

UCASE function 7–24, A–27

Undefined code page
definition of 2–11

Unicode
advantages of 9–3
definition of 9–2
designing queries for 9–8
fonts for 9–5, 9–10
Progress support for 1–5

UNIX 2–9

UPDATE statement 8–14

UPPER function 7–24, A–27

USE_IT
word-delimiter attribute B–10

User-defined characters 8–33

UTF-8 3–12

V

VARCHAR data type 7–14, A–17

Visual Translator 1–6

W

WBREAK-COMPILER qualifier
PROUTIL utility A–32

Windows screen and keyboard
code-page of 2–15

Word-break behavior 8–36

Word-break tables 3–21
associating with databases 3–28
compiling 3–27
creating and modifying 3–22
providing access to 3–27
rationale 3–21
syntax 3–25, B–8

Word-delimiter attributes 3–22

WORDRULES qualifier
PROUTIL utility A–32

Y

YEAR function 4–11

-yy startup parameter A–9