Progress®



Internationalize ABL Applications

OpenEdge®

Copyright

© **2019 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.**

These materials and all Progress® software products are copyrighted and all rights are reserved by Progress Software Corporation. The information in these materials is subject to change without notice, and Progress Software Corporation assumes no responsibility for any errors that may appear therein. The references in these materials to specific platforms supported are subject to change.

Corticon, DataDirect (and design), DataDirect Cloud, DataDirect Connect, DataDirect Connect64, DataDirect XML Converters, DataDirect XQuery, DataRPM, Deliver More Than Expected, Icenium, Kendo UI, Kinvey, NativeScript, OpenEdge, Powered by Progress, Progress, Progress Software Developers Network, Rollbase, SequeLink, Sitefinity (and Design), Sitefinity, SpeedScript, Stylus Studio, TeamPulse, Telerik, Telerik (and Design), Test Studio, and WebSpeed are registered trademarks of Progress Software Corporation or one of its affiliates or subsidiaries in the U.S. and/or other countries. Analytics360, AppServer, BusinessEdge, DataDirect Autonomous REST Connector, DataDirect Spy, SupportLink, DevCraft, Fiddler, JustAssembly, JustDecompile, JustMock, Kinvey Chat, NativeChat, NativeScript Sidekick, OpenAccess, ProDataSet, Progress Results, Progress Software, ProVision, PSE Pro, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, and WebClient are trademarks or service marks of Progress Software Corporation and/or its subsidiaries or affiliates in the U.S. and other countries. Java is a registered trademark of Oracle and/or its affiliates. Any other marks contained herein may be trademarks of their respective owners.

February 2019

Last updated with new content: Release 12.0

Updated: 2019/03/01

Table of Contents

Preface....11
Purpose....11
Audience....11
Organization....12
Using ABL documentation....13
References to ABL compiler and run-time features....13
References to ABL data types....13
Typographical conventions....13
Examples of syntax descriptions....15
Long syntax descriptions split across lines....16
Complex syntax descriptions with both required and optional elements....17
Examples of SQL syntax diagrams....17
Long SQL syntax descriptions split across lines....18
OpenEdge messages....18
Obtaining more information about OpenEdge messages....19

Chapter 1: Developing Applications for Deployment Worldwide....21
The importance of internationalization and localization....21
Strategies for responding to a global market....22
Internationalized applications....22
Localized applications....22
How OpenEdge supports internationalization....23
Multi-byte applications....23
OpenEdge support for Unicode....23
Character processing....23
OpenEdge ABL elements....24
International databases....24
OpenEdge messages....24
Regional parameter files....25
How OpenEdge products support internationalization....25
Design guidelines....26

Chapter 2: Understanding Code Pages....27
Code pages and character sets....27
Code pages....28
Character sets....30
Environments with multiple code pages....32
Code-page conversion....33

Streams and code-page conversion....34
Valid and invalid code-page conversions....34
Sockets and code-page conversion....35
The undefined code page....35
Determining which code page an application component uses....36
ASCII character data....36
8-bit character data....36
OpenEdge databases....37
Character terminals....37
Windows screen and keyboard....38
Printers....38
Table dump (.d) files and other external text files....39
OpenEdge libraries....39

Chapter 3: Understanding Character Processing Tables....41

The convmap.dat file and its tables....42
Character attribute tables....43
Case tables....44
Collations and collation tables....46
Code-page conversion tables....47
Modifying convmap.dat or a file it includes....49
Compiling the convmap.dat file....49
Providing access to the convmap.cp file....50
Modifying OpenEdge collation tables....50
Finding additional information on character processing tables other than word-break tables....53
Word-break tables....53
Why OpenEdge uses word-break tables....53
Creating and modifying word-break tables....54
Compiling word-break tables....57
Providing access to word-break tables....57
Associating word-break tables with databases....58
Rebuilding the indexes for word-break table changes....58

Chapter 4: Preparing the Code....61

Guidelines and methodology....61
Structuring source code....62
Processing by characters....62
Using variables....63
Coding with translation in mind....63
Input and output....64
Keyboards....64
Printers....64
Data-processing issues....65

Numeric formats....65
Measurements....66
Currencies....66
Dates....67
Time....68
Addresses....68
Sorting data....69
Local conventions....69
Using collation tables....69
Using case tables....71

Chapter 5: Preparing the User Interface....73

Screen layout and composition....74
Equipment differences....74
Monitors....74
Keyboards....74
Culturally specific issues....74
Graphics and icons....78
Text in graphics and icons....78
Images to avoid in graphics and icons....78
Designing with the AppBuilder....80
Localizing SmartObject masters....80
Localizing SmartObject instances....80
Using multiple layouts....82
Designing to allow for translation....83
Text expansion and contraction....83
Message text....84
Layout....84

Chapter 6: Using Databases....85

OpenEdge database name restrictions....85
Empty databases....86
Empty databases supplied by OpenEdge....86
Creating an empty database in a particular code page....86
Scanning databases for character conflicts....87
New characters....87
Using PROUTIL to perform the scan....89
Converting a database and its data to a different code page....90
Loading table dump files....91
Code-page trailers....91
Techniques for loading table dump files....92
Specifying -cpinternal and -cpstream with database utilities....92

Chapter 7: Using SQL....93

Starting database servers....93
Using SQL database servers....95
Code-page conversion in SQL applications....95
Using SQL clients....97
Java/JDBC clients....98
ODBC clients....98
Characteristics of OpenEdge SQL clients....98
Using SQL utilities....98
SQLDUMP....99
SQLLOAD....100
SQLSCHEMA....101
Using the SQL language....102
Unit of length when working with character strings....102
Maximum bytes required for CHAR and VARCHAR....103

Chapter 8: Using Multi-byte Code Pages....105

Definitions of key terms....105
Terms for code pages....106
Terms for characters....106
Terms for bytes....106
OpenEdge support....107
Input....108
Using the keyboard and mouse....109
Using ABL....110
Automatic input validation....112
Output....112
FORMAT phrase....112
Printing....114
Character-client color limit....115
Inside the multi-byte application....115
Distinguishing characters, bytes, and columns....115
Techniques for working with multi-byte characters....116
ABL support for processing multi-byte characters....120
Issues specific to multi-byte code pages....120
OpenEdge support for multi-byte code pages....120
Valid and invalid code-page conversions....122
User-defined characters....123
Collating multi-byte characters....124
Default word-break behavior of characters in multi-byte code pages....125
Guidelines for using multi-byte characters....126

Chapter 9: Using Unicode....127

Unicode overview....128
Why use Unicode....128
The limits of multiple code pages....128
The advantages of Unicode....128
Using Unicode with OpenEdge products....129
Using Unicode with OpenEdge databases....130
Converting an OpenEdge database to UTF-8 using the PROUTIL CONVCHAR utility....130
Converting an OpenEdge database to UTF-8 using dump and load....131
Compiling, storing, and applying the UTF-8 word-break rules to a database....132
Using Unicode with OpenEdge applications....132
Rules for using Unicode with the OpenEdge character client....132
Rules for using Unicode with the OpenEdge graphical client....133
Unicode application example....133
Guidelines for using Unicode....134
Unicode support for supplementary characters....135
Using UTF-16 in the ASC and CHR functions....135
Behavior of ASC and CHR functions....136
New and modified keywords....137
Limitations....137
ABL constructs that support Unicode filenames....137

Chapter 10: Deployment and Configuration....143

The progress.ini file and the Windows registry....144
[Startup]....144
[fonts]....145
Specifying scripts....145
The PROTERMCAP file....146
The PROMSGS file....146
Parameter files....147
The conmgr.properties file....148
Report Builder parameters....149
Additional guidelines....150
Using the UNIX character client with double-byte code pages....150
Specifying -cpinternal and -cpstream with database utilities....150

Appendix A: OpenEdge Resources....151

Files in the OpenEdge/prolang directory....151
Startup parameters and settings....152
ABL....155
OpenEdge SQL....160
SQL elements that support internationalization and localization....160

Format specifiers allowed with the TO_CHAR and TO_DATE functions....167
Utilities....170
Determining the code page....171

Appendix B: Character Processing Table Formats....175

Character attribute table....176
Case table....177
Collation table....178
Code-page conversion table....179
Convmap changes....180
Word-break table....181

Index....185

Preface

For details, see the following topics:

- Purpose
- Audience
- Organization
- Using ABL documentation
- Typographical conventions
- Examples of syntax descriptions
- Examples of SQL syntax diagrams
- OpenEdge messages

Purpose

This book provides information about developing and deploying OpenEdge® applications for a worldwide market. It describes important considerations when designing and developing business applications for international deployment.

Audience

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

For the latest documentation updates go to https://docs.progress.com.

Organization

Developing Applications for Deployment Worldwide on page 21

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

Understanding Code Pages on page 27

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

Understanding Character Processing Tables on page 41

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

Preparing the Code on page 61

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

Preparing the User Interface on page 73

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

Using Databases on page 85

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

Using SQL on page 93

Describes how to use SQL in applications to be deployed across multiple locales.

Using Multi-byte Code Pages on page 105

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

Using Unicode on page 127

Provides information on OpenEdge support for multi-lingual applications through implementation of the Unicode Standard character set.

Deployment and Configuration on page 143

Provides guidelines for deploying and configuring international applications.

OpenEdge Resources on page 151

Lists the OpenEdge resources that support international development. Describes resources at the system level, the ABL level, the SQL level, and the utility level.

Character Processing Table Formats on page 175

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

Using ABL documentation

OpenEdge provides a special purpose programming language for building business applications. In the documentation, the formal name for this language is *ABL (Advanced Business Language)*. With few exceptions, all keywords of the language appear in all `UPPERCASE`, using a font that is appropriate to the context. All other alphabetic language content appears in mixed case.

For the latest documentation updates go to https://docs.progress.com.

References to ABL compiler and run-time features

ABL is both a compiled and an interpreted language that executes in a run-time engine. The documentation refers to this run-time engine as the *ABL Virtual Machine (AVM)*. When the documentation refers to ABL source code compilation, it specifies *ABL* or *the compiler* as the actor that manages compile-time features of the language. When the documentation refers to run-time behavior in an executing ABL program, it specifies *the AVM* as the actor that manages the specified run-time behavior in the program.

For example, these sentences refer to the ABL compiler's allowance for parameter passing and the AVM's possible response to that parameter passing at run time: "ABL allows you to pass a dynamic temp-table handle as a static temp-table parameter of a method. However, if at run time the passed dynamic temp-table schema does not match the schema of the static temp-table parameter, the AVM raises an error." The following sentence refers to run-time actions that the AVM can perform using a particular ABL feature: "The ABL socket object handle allows the AVM to connect with other ABL and non-ABL sessions using TCP/IP sockets."

References to ABL data types

ABL provides built-in data types, built-in class data types, and user-defined class data types. References to built-in data types follow these rules:

- Like most other keywords, references to specific built-in data types appear in all `UPPERCASE`, using a font that is appropriate to the context. No uppercase reference ever includes or implies any data type other than itself.
- Wherever *integer* appears, this is a reference to the `INTEGER` or `INT64` data type.
- Wherever *character* appears, this is a reference to the `CHARACTER`, `LONGCHAR`, or `CLOB` data type.
- Wherever *decimal* appears, this is a reference to the `DECIMAL` data type.
- Wherever *numeric* appears, this is a reference to the `INTEGER`, `INT64`, or `DECIMAL` data type.

References to built-in class data types appear in mixed case with initial caps, for example, `Progress.Lang.Object`. References to user-defined class data types appear in mixed case, as specified for a given application example.

Typographical conventions

This documentation uses the following typographical and syntax conventions:

Convention	Description
Bold	Bold typeface indicates commands or characters the user types, provides emphasis, or the names of user interface elements.
Italic	Italic typeface indicates the title of a document, or signifies new terms.
SMALL, BOLD CAPITAL LETTERS	Small, bold capital letters indicate OpenEdge key functions and generic keyboard keys; for example, **GET** and **CTRL**.
KEY1+KEY2	A plus sign between key names indicates a **simultaneous** key sequence: you press and hold down the first key while pressing the second key. For example, **CTRL+X**.
KEY1 KEY2	A space between key names indicates a **sequential** key sequence: you press and release the first key, then press another key. For example, **ESCAPE H**.
Syntax:	
`Fixed width`	A fixed-width font is used in syntax, code examples, system output, and file names.
`Fixed-width italics`	Fixed-width italics indicate variables in syntax.
`Fixed-width bold`	Fixed-width bold italic indicates variables in syntax with special emphasis.
`UPPERCASE fixed width`	ABL keywords in syntax and code examples are almost always shown in upper case. Although shown in uppercase, you can type ABL keywords in either uppercase or lowercase in a procedure or class.
Period (.) or colon (:)	All statements except `DO`, `FOR`, `FUNCTION`, `PROCEDURE`, and `REPEAT` end with a period. `DO`, `FOR`, `FUNCTION`, `PROCEDURE`, and `REPEAT` statements can end with either a period or a colon.
[ ]	Large brackets indicate the items within them are optional.
[ ]	Small brackets are part of ABL.
{ }	Large braces indicate the items within them are required. They are used to simplify complex syntax diagrams.
{ }	Small braces are part of ABL. For example, a called external procedure must use braces when referencing arguments passed by a calling procedure.
|	A vertical bar indicates a choice.
. . .	Ellipses indicate repetition: you can choose one or more of the preceding items.

Examples of syntax descriptions

In this example, ACCUM is a keyword, and *aggregate* and *expression* are variables:

Syntax

```
ACCUM aggregate expression
```

FOR is one of the statements that can end with either a period or a colon, as in this example:

```
FOR EACH Customer NO-LOCK:
  DISPLAY Customer.Name.
END.
```

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

Syntax

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

In this example, the outer (small) brackets are part of the language, and the inner (large) brackets denote an optional item:

Syntax

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

A called external procedure must use braces when referencing compile-time arguments passed by a calling procedure, as shown in this example:

Syntax

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

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

Syntax

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

In this example, you must include two expressions, and optionally you can include more. Multiple expressions are separated by commas:

Syntax

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

In this example, you must specify MESSAGE and at least one *expression* or SKIP [(*n*)], and any number of additional *expression* or SKIP [(*n*)] 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" ] ... }
```

Long syntax descriptions split across lines

Some syntax descriptions are too long to fit on one line. When syntax descriptions are split across multiple lines, groups of optional and groups of required items are kept together in the required order.

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

Syntax

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

Complex syntax descriptions with both required and optional elements

Some syntax descriptions are too complex to distinguish required and optional elements by bracketing only the optional elements. For such syntax, the descriptions include both braces (for required elements) and brackets (for optional elements).

In this example, ASSIGN requires either one or more *field* entries or one *record*. Options available with *field* or *record* are grouped with braces and brackets:

Syntax

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

Examples of SQL syntax diagrams

In this example, GRANT, RESOURCE, DBA, and TO are keywords. You must specify RESOURCE, DBA, or both, and at least one *user_name*. Optionally you can specify additional *user_name* items; each subsequent *user_name* must be preceded by a comma:

Syntax

```
GRANT { RESOURCE, DBA } TO user_name[, user_name] . . . ;
```

This excerpt from an ODBC application invokes a stored procedure using the ODBC syntax { call *procedure_name (param)* }, where braces and parentheses are part of the language:

Syntax

```
proc1( param, "{ call proc2 (param) }", param);
```

In this example, you must specify a *table_name*, *view_name*, or *synonym*, but you can choose only one. In all SQL syntax, if you specify the optional *owner_name* qualifier, there must not be a space between the period separator and *table_name*, *view_name*, or *synonym*:

Syntax

```
CREATE [ PUBLIC ] SYNONYM synonym
  FOR [ owner_name.] {table_name | view_name | synonym} ;
```

In this example, you must specify *table_name* or *view_name*:

Syntax

```
DELETE FROM [owner_name.] {table_name | view_name}
  [ WHERE search_condition] ;
```

In this example, you must include one expression *(expr)* or column position *(posn)*, and optionally you can specify the sort order as ascending (ASC) or descending (DESC). You can specify additional expressions or column positions for sorting within a sorted result set. The SQL engine orders the rows on the basis of the first *expr* or *posn*. If the values are the same, the second *expr* or *posn* is used in the ordering:

Syntax

```
ORDER BY {expr | posn} [ ASC | DESC ]
  [ , [{expr | posn} [ ASC | DESC ]] . . . ]
```

Long SQL syntax descriptions split across lines

Some syntax descriptions are too long to fit on one line. When syntax descriptions are split across multiple lines, groups of optional and groups of required items are kept together in the required order.

In this example, CREATE VIEW is followed by several optional items:

Syntax

```
CREATE VIEW [owner_name.] view_name
  [ ( column_name[, column_name] . . . ) ]
  AS [ ( ] query_expression[ ) ] [ WITH CHECK OPTION ] ;
```

OpenEdge messages

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

- **Execution messages** inform you of errors encountered while OpenEdge is running a procedure; for example, if OpenEdge cannot find a record with a specified index field value.
- **Compile messages** inform you of errors found while OpenEdge is reading and analyzing a procedure before 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 OpenEdge is getting ready to execute; for example, if you entered an invalid startup parameter.

After displaying a message, OpenEdge 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 after execution messages.
- Returns to the Procedure Editor, so you can correct an error in a procedure. This is the usual action taken after compiler messages.
- Halts processing of a procedure and returns immediately to the Procedure Editor. This does not happen often.
- Terminates the current session.

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

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

If you encounter an error that terminates OpenEdge, note the message number before restarting.

Obtaining more information about OpenEdge messages

In Windows platforms, use OpenEdge online help to obtain more information about OpenEdge messages. Many OpenEdge tools include the following Help menu options to provide information about messages:

- Choose **Help** > **Recent Messages** to display detailed descriptions of the most recent OpenEdge message and all other messages returned in the current session.
- Choose **Help** > **Messages** and then type the message number to display a description of a specific OpenEdge message.
- In the Procedure Editor, press the **HELP** key or **F1**.

On UNIX platforms, use the OpenEdge `pro` command to start a single-user mode character OpenEdge client session and view a brief description of a message by providing its number.

To use the pro command to obtain a message description by message number:

1. Start the Procedure Editor:

   ```
   OpenEdge-install-dir/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**.

1

Developing Applications for Deployment Worldwide

This chapter summarizes issues connected with developing OpenEdge® applications for deployment worldwide.

For details, see the following topics:

- The importance of internationalization and localization
- Strategies for responding to a global market
- How OpenEdge supports internationalization
- How OpenEdge products support internationalization
- Design guidelines

The importance of internationalization and localization

Imagine that you just bought an imported sound system. You open the box, find an instruction booklet written in your language, and begin reading, but you soon notice:

- The language of the booklet sounds stilted.
- The punctuation and capitalization do not use the rules of your language.
- In the section on basic setup, dates and times are not formatted the way you are used to.
- Some of the illustrations contain text that is not translated into a language you understand.

You you may worry that the sound system 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.

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.

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.

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.

How OpenEdge supports internationalization

All components of OpenEdge support your international development efforts, including the Application Development Environment (ADE) tools, ABL, and the database. In addition, as part of the installation media, OpenEdge provides files and templates 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 OpenEdge to develop and deploy applications worldwide.

OpenEdge 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.

Multi-byte applications

OpenEdge 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 Using Multi-byte Code Pages on page 105. For information on running multi-byte character clients, see Deployment and Configuration on page 143.

OpenEdge support for Unicode

OpenEdge supports multi-lingual 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 Using Unicode on page 127.

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. OpenEdge supports a wide range of character sets and code pages that accommodate many languages. The OpenEdge database and application environment have administrative options that allow you to manage code pages and control where conversions take place. See Understanding Code Pages on page 27 and Understanding Character Processing Tables on page 41 for a discussion of the issues and for instructions on how to manage code pages for your databases and applications.

OpenEdge ABL elements

OpenEdge 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 OpenEdge fully support multi-byte character sets. See OpenEdge Resources on page 151 for a list of OpenEdge elements that support internationalization.

International databases

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

These empty databases provide a database labeled 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 `OpenEdge/prolang`, OpenEdge has a utility, `PROUTIL`, that you can use to set up a unique database. For more information on `PROUTIL`, see *Manage the OpenEdge Database*.

OpenEdge messages

The text used in OpenEdge messages is contained in the `promsgs` file. OpenEdge provides various language editions of the `promsgs` file in the `OpenEdge/prolang` subdirectories. Each file has an extension that identifies its language. The following excerpt of the Dutch version of OpenEdge messages is from `OpenEdge/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 OpenEdge with a certain version of `promsgs`, set the `PROMSGS` environment variable to the appropriate file location. By default, OpenEdge retrieves messages from `OpenEdge/promsgs`.You can also change the `PROMSGS` language dynamically, from within your application.

Regional parameter files

A useful technique for controlling an OpenEdge client session or server is to use a parameter (`.pf`) file with a startup or connection command. OpenEdge provides parameter files that set up OpenEdge 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 OpenEdge 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 OpenEdge Translation Management System (`-lng "Swedish"`). OpenEdge 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`).

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

How OpenEdge products support internationalization

The OpenEdge database supports, among other code pages, the Unicode Standard character set, which simplifies multi-national deployment. See Using Unicode on page 127 for information on Unicode. See *Introduction to the Database* for a discussion of designing and creating effective databases.

The AppServer™ provides support for multiple code pages and languages. The AppServer supports the same code-page options as the ABL 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 ABL client.

The Open Client Toolkit supports international Java™ and ActiveX Controller client applications by communicating with the AppServer using Unicode. The AppServer then converts between Unicode and the internal code page defined for the AppServer. See Using Unicode on page 127 for information on Unicode and UTF-8.

OpenEdge DataServers for Oracle and Microsoft SQL Server support international applications. Configuring these products for an international locale involves configuring both the OpenEdge environment and the non-OpenEdge 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 *Use the Oracle Data Server*or *Use the Microsoft SQL Server DataServer* for information on managing code pages and converting between these environments.

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 multi-lingual support and if the OpenEdge implementation of the Unicode Standard meets your requirements

For more information on design guidelines, see Preparing the User Interface on page 73.

For more information on programming and system administration issues that you must consider when developing and deploying an international OpenEdge application, see Preparing the Code on page 61 and Deployment and Configuration on page 143.

For more information on issues that affect applications that use multi-byte character sets, see Using Multi-byte Code Pages on page 105 and Using Unicode on page 127.

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 code page table that associates each numeric value with the corresponding character. Code pages play key roles in applications to be adapted to multiple locales.

For details, see the following topics:

- Code pages and character sets
- Environments with multiple code pages
- Code-page conversion
- The undefined code page
- Determining which code page an application component uses

Code pages and character sets

Code pages and character sets are not the same.

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: no two characters are assigned the same numeric value, and no two numeric values are assigned the same character.

The following figure 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 shaded areas contain the numeric values. By convention, numeric values are in hexadecimal. 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.

Figure 1: The IBM850 code page

	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)

The following figure shows the characters and numeric values of the ISO8859-1 code page, also widely used in western Europe and the Americas.

Figure 2: The ISO8859-1 code page

	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
0		0	@	P	,	p			(NSP)	°	À	Ð	â	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

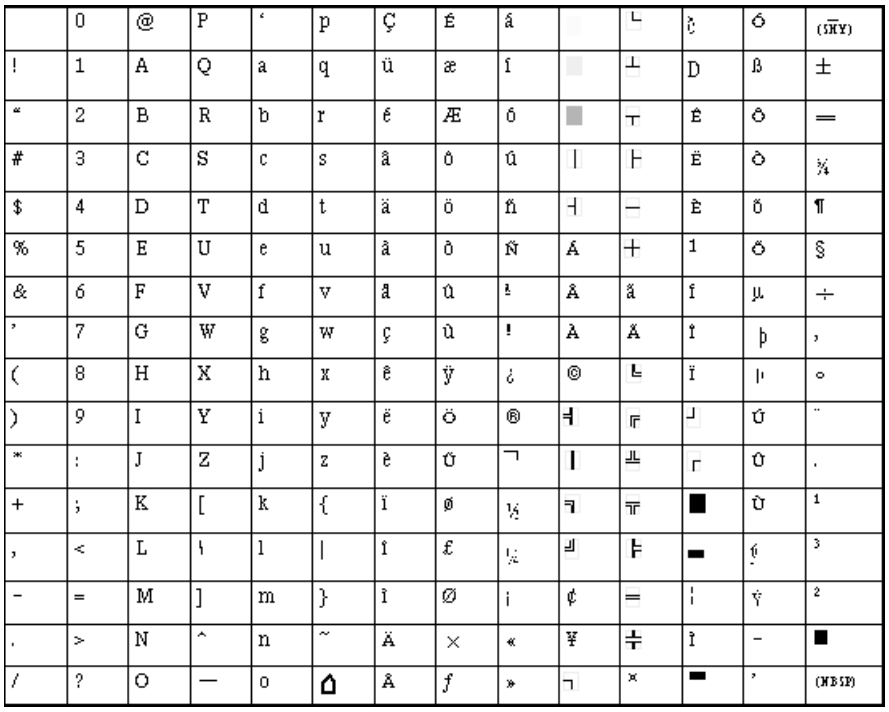
Here are some other important points about code pages:

- 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 multi-byte (1-4 bytes), 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 multi-byte code page is UTF-8, an encoding of Unicode. For more information on using multi-byte code pages in applications, see Using Multi-byte Code Pages on page 105 and Using Unicode on page 127.

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. The following figure shows the character set of the IBM850 code page.

Figure 3: The character set of the IBM850 code page



The following figure shows the character set of the ISO8859-1 code page.

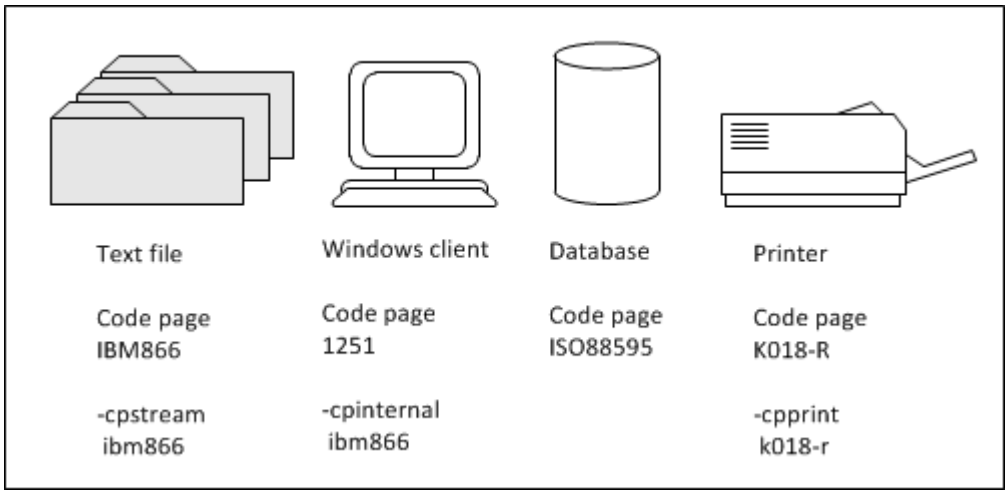
Figure 4: The character set of the ISO8859-1 code page

	0	@	P	,	p			(RSP)	°	À	Ð	à	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

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. The following figure shows a Russian application, where each component has a different code page. The figure also shows startup parameters, each of which tells OpenEdge the code page of the corresponding component. For more information on using startup parameters, see Deployment and Configuration on page 143.

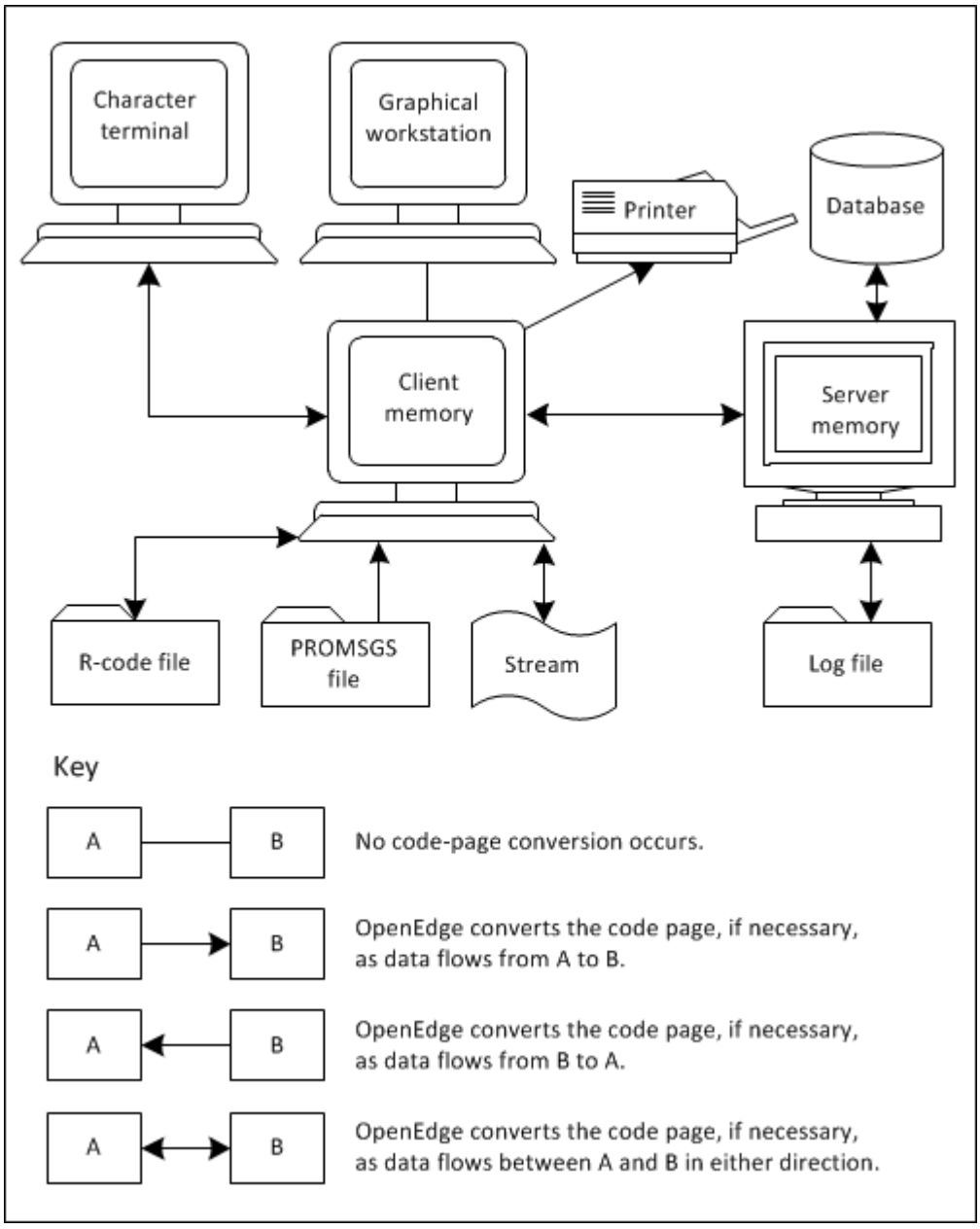
Figure 5: A Russian application where each component has a different code page



Code-page conversion

As data flows between application components that have different code pages, OpenEdge sometimes converts the data from the code page of the source component to the code page of the destination component. This is illustrated in the following figure.

Figure 6: Code-page conversion



For each line in the figure, OpenEdge 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, OpenEdge uses a code-page conversion table. OpenEdge supplies a collection of code-page conversion tables in the `OpenEdge/prolang` directory. You can also create your own code-page conversion tables. For more information on code-page conversion tables, see Understanding Character Processing Tables on page 41. For more information on code-page conversion and databases, see Using Databases on page 85. For more information on code-page conversion and SQL, see Using SQL on page 93.

Streams and code-page conversion

The previous figure 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 an OpenEdge application reads or writes
- `LISTING`, `XREF`, and `PREPROCESS` files produced by the compiler

As the figure in Code-page conversion on page 33 shows, to decide whether or not to convert the code page of data flowing between two application components, OpenEdge considers whether one of the application components is a stream.

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 OpenEdge 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 *Manage ABL Applications*.

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 Using Databases on page 85.

Sockets and code-page conversion

If two applications using different code pages communicate using sockets (one of the external programming interfaces OpenEdge 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 ABL `CODEPAGE-CONVERT` function either before invoking the ABL `PUT-STRING` function or after invoking the ABL `GET-STRING` function.

The undefined code page

Caution: Any use of the undefined code page runs the risk of corrupting non-ASCII data.

A special code-page name, undefined, tells OpenEdge 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 OpenEdge 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 Using Databases on page 85

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

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.

Determining which code page an application component uses

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

ASCII character data

ASCII characters are numbered 1-127 and can be represented with seven bits. If you are certain some character data contains only ASCII characters and you know the code page of your current environment, you need to place only the following commands into `OpenEdge/startup.pf`, the parameter file every local OpenEdge executable reads at startup:

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

codepage-name

The code page of your current environment.

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 `OpenEdge/prolang` directory) to examine the contents of the file. Before you run this procedure, make sure you start OpenEdge 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 below for the following data locations:

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

- OpenEdge libraries

OpenEdge databases

There are several options that can be used to determine the code page of an OpenEdge database, and some are described in this section.

To determine the code page of an OpenEdge database:

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 **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 option 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 option is to use the `DBCODEPAGE` function in a procedure, as shown in the following code fragment:

```
DEFINE VARIABLE ix AS INTEGER NO-UNDO.
REPEAT ix = 1 TO NUM-DBS:  /* For every connected database */
  DISPLAY DBCODEPAGE(ix).
END.
```

For more information on the `DBCODEPAGE` function, see *ABL Reference*.

Character terminals

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

- 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 options 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, OpenEdge does not see the code page of the operating system, but rather the new one. You must determine which code page OpenEdge actually sees.

To determine which code page OpenEdge actually sees:

1. Start OpenEdge 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, OpenEdge 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 OpenEdge displays is the 8-bit value for that character.
4. Once you have several such pairs of characters and numbers, compare them against likely code pages to determine the correct code page.

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:

1. In 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 corresponding to the three-digit decimal number appears on the screen.
4. Repeat Steps 2 and 3 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 OpenEdge on Windows, set `-cpinternal` to `undefined`, and run the following code fragment:

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

Printers

To determine the code page a printer uses, check the documentation that comes with both 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 OpenEdge with `-cpprint` set to `undefined`, then run the following code fragment:

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

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

Table dump (.d) files and other external text files

If you know which 8-bit characters your data contains and you 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 OpenEdge ABL program that has the following syntax:

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

filename

The name of your file.

Note: Start OpenEdge with `-cpinternal` set to `undefined`.

OpenEdge libraries

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

```
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 *Manage ABL Applications*.

3

Understanding Character Processing Tables

OpenEdge 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. OpenEdge 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 `OpenEdge/prolang/convmap` directory. You can create a word-break table from scratch or modify one that OpenEdge 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.

For a thorough explanation of code pages, see Understanding Code Pages on page 27.

For details, see the following topics:

- The convmap.dat file and its tables
- Word-break tables

The convmap.dat file and its tables

The `convmap.dat` file supplied by OpenEdge, which resides in the `OpenEdge/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 `OpenEdge/prolang/convmap` directory.

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. The following table lists the contents of `arabic.dat`.

Table 1: Contents of arabic.dat

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
711	BASIC case table BASIC collation table Character attribute table Table for converting from code page 1256 to code page 711

Code page	Character processing tables
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

The table 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 the table shows.
- Every code page does not have every kind of character processing table

Character attribute tables

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

The following figure shows code page 1256's character attribute table, the first table in the `arabic.dat` file.

Code page 1256's character attribute table

```
# 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
```

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. The figure shows that elements 67, 210, and 250 are alphabetic, while elements 48, 63, and 238 are nonalphabetic.

Case tables

A case table tells OpenEdge how to convert a character in the code page from uppercase to lowercase or from lowercase to uppercase. OpenEdge uses a case table when it encounters code such as the OpenEdge ABL `CAPS` and `LC` functions.

The following figure shows code page ISO8859-15's BASIC case table, which resides in the `8859-15.dat` file.

Code page ISO8859-15's BASIC case table

```
# 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
```

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. The figure 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.

Collations and collation tables

A collation is a set of rules that determine how character data is ordered. OpenEdge provides a set of collation tables to sort and compare character data. OpenEdge also supports the International Components for Unicode (ICU) collations, which provide linguistic sorting of Unicode data based on the Unicode Collation Algorithm. OpenEdge uses a collation when it does the following:

- Compares `CHARACTER` strings using a ABL relational operator or the ABL `COMPARE` function
- Generates the collation value of a `CHARACTER` data item using the ABL `COLLATE` option of the `FOR` statement, the `OPEN QUERY` statement, and the `PRESELECT` phrase
- Sorts the results of a ABL query that uses the FOR statement's `BY` or `EACH` option
- Builds or rebuilds databases indexes

Use the Collation Table (`-cpcoll`) startup parameter to specify the ICU collation or OpenEdge collation table you want use. You can also specify an OpenEdge collation or an ICU collation with the ABL `COMPARE` function and the `COLLATE` option on the ABL `FOR` statement, `OPEN QUERY` statement, and `PRESELECT` phrase.

Note: For more information about character processing and ICU collations, refer to the README.txt file in the directory OpenEdge/prolang.

The following figure shows code page 1253's GREEK collation table, which resides in the `greek.dat` file.

Code page 1253's GREEK collation table

```
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
```

The figure 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.

Code-page conversion tables

A code-page conversion table tells OpenEdge how to convert a character on one code page to the equivalent character on another code page. OpenEdge 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 Using Databases on page 85.
- OpenEdge automatically converts data from one code page to another during execution. For more information on automatic code-page conversion, see Understanding Code Pages on page 27.

The following figure 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 `arabic.dat` file.

Table for converting from code page 1256 to code page 709

```
# 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
```

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 the figure 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 , 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 shown in the following table.

Table 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, OpenEdge automatically computes the inverse conversions and assigns the correct value for `TYPE`, as shown in the following table.

Table 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 Using Multi-byte Code Pages on page 105 and Using Unicode on page 127.

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 Character Processing Table Formats on page 175.

Note: Modifying collation tables involves additional steps. For more information on modifying collation tables, see Modifying OpenEdge collation tables on page 50.

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` qualifier. The syntax is:

Syntax

```
proutil -C CODEPAGE-COMPILER sourcefile.datoutputfile.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 *Manage the OpenEdge Database*.

Providing access to the convmap.cp file

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

- Copy the convmap.cp file to the OpenEdge installation directory
- Set the PROCONV environment variable to point to the convmap.cp file
- Use the conversion map (-convmap) startup parameter, for example:

```
-convmap convmap-file-pathname
```

Modifying OpenEdge collation tables

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

Note: If you need to modify an International Components for Unicode (ICU) collation, contact Progress Software Technical Support for assistance.

Preliminary considerations

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

To modify how ABL comparisons are performed, start OpenEdge against the database that contains the collation tables you want, or use the collation table (-cpcoll) startup parameter to point to the collation tables you want. When you specify a collation table using the -cpcoll startup parameter, you 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 Editing the collation table on page 51.

Dumping a collation table

Dumping a collation table is the first step in modifying the table.

To dump a collation table:

1. In the Data Administration tool, choose **Admin** > **Dump Data and Definitions** > **Collation**. The **Dump Collation Tables** dialog box appears.
2. In the **Output File** field, type a dump name (`_tran.df` is the default), then choose **OK**.
3. Enter the name of the code page with which the file will be written out.

The `_tran.df` file contains the collation table for your database. The following figure shows a sample `_tran.df` file.

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
```

Note: The IBM850 code page that appears in the trailer of the sample `_tran.df` file in the figure 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, refer to the Greek collation table in Collations and collation tables on page 46.

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 OpenEdge that the following table entry is for ABL 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, OpenEdge searches the convmap.cp file (or an equivalent file) to locate the correct collation tables. As a key to these tables, OpenEdge 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 OpenEdge uses internally. Progress Software Corporation recommends 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 Modifying OpenEdge collation tables on page 50.

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

Loading the modified collation table

Loading the modified collation table is the next step in modifying the table.

To load the modified table back into the database:

1. In the Data Administration tool, choose **Admin** > **Load Data and Definitions** > **Data Definitions (.df file)**. The **Load Data Definitions** dialog box appears.
2. In the **Input File** field, type the name of the data definition file you modified (the default name is _tran.df), then choose **OK**.

Rebuilding the indexes for collation table changes

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

The database cannot be in use when you do this.

The minimum syntax of the PROUTIL utility with the IDXBUILD qualifier to rebuild all indexes is:

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

For the complete syntax of the PROUTIL utility, see *Manage the OpenEdge Database*.

Modifying, compiling, and providing access to the CONVMAP file

You must still perform the following tasks:

- Modify the convmap.dat file. For more information, see Modifying convmap.dat or a file it includes on page 49.
- Compile the convmap.dat file. For more information, see Compiling the convmap.dat file on page 49
- Provide access to the resulting convmap.cp file. For more information, see Providing access to the convmap.cp file on page 50.

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 Deployment and Configuration on page 143 for information on setting startup parameters to tell OpenEdge which code page, collation table, or case table to use
- Run the procedure `list-cp.p` in the `OpenEdge/prolang` directory to see a list of the collations OpenEdge supports
- Run the procedure `listconv.p` in the `OpenEdge/prolang` directory to see a list of the code-page conversions OpenEdge supports
- See the `readme` file in the `OpenEdge/prolang` directory for information on:
  - The locale-specific subdirectories of the `OpenEdge/prolang` directory
  - The case tables OpenEdge provides
  - The collation tables OpenEdge provides
  - The code-page conversion tables OpenEdge provides

Word-break tables

OpenEdge uses word-break tables to process ABL queries that use the `CONTAINS` operator of the `WHERE` option of the record phrase.

The following example shows a ABL query that uses `CONTAINS`:

```
FOR EACH customer NO-LOCK WHERE customer.comments CONTAINS "credit hold":
  DISPLAY customer.name customer.custnum customer.comments.
END.
```

Why OpenEdge uses word-break tables

OpenEdge 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, OpenEdge must quickly find the database records that contain the words of the target string (for example, "credit" and "hold"). To accelerate its searching, OpenEdge uses word indexes. But before word indexes can be used, they must be built. To build them, OpenEdge breaks each database field to be word indexed into separate words. To do this correctly, OpenEdge 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, and so on), while an e-mail application might consider these characters to be parts of words—in this case, e-mail addresses.

Creating and modifying word-break tables

OpenEdge provides a collection of word-break tables in the `OpenEdge/prolang/convmap` directory. the following figure shows one of them, `big5-bas.wbt`. This filename reflects the code page big-5, a code page used for Traditional Chinese.

The `big5-bas.wbt` word-break table

```
/* NAME: big5-bas.wbt
 * OpenEdge 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
};
```

Understanding word-delimiter attributes

The keywords `BEFORE_DIGIT`, `IGNORE`, and `USE_IT`, which appear in , 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 the following table.

Table 4: Word-delimiter attributes

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	-
BEFORE_LETTER	Part of a word only if followed by a character with the LETTER attribute, otherwise treated as a word delimiter.	-
DIGIT	Always part of a word	Assigned to the characters 0-9.
IGNORE	Ignored	Assigned to the apostrophe (').For example, "John's" is equivalent to "Johns."
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.
TERMINATOR	Word delimiter	Assigned to all other characters.
USE_IT	Always part of a word	Assigned to the following characters: • Dollar sign ($) • Percent sign (%) • Number sign (#) • At symbol (@) • Underline (_)

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 OpenEdge provides and modify it as necessary. You can also create a word-break table from scratch. The syntax is as follows:

Syntax

```
[ #define symbolic-namesymbol-value ] ...

[ Version = 9
   Codepage = codepage-name
   wordrules-name = wordrules-name
   type = 3
]

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 OpenEdge and some versions of OpenEdge let you compile word-break tables that omit all items within the second pair of square brackets, Progress Software Corporation recommends 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, Progress Software Corporation recommends 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.

type=3

The table type. Although OpenEdge supports existing word-break tables of type 1 and type 2, Progress Software Corporation recommends you change their table type to 3. If you do, you also must 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 the table.

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.

Syntax

```
proutil -C wbreak-compiler src-filerule-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 OpenEdge 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 *Manage the OpenEdge Database*.

Providing access to word-break tables

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

- Copy the compiled word-break table to the OpenEdge 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

OpenEdge/mydir/mysubdir directory, set the environment variable PROWD34 to OpenEdge/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.

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.

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.

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 *Manage the OpenEdge Database*.

Rebuilding the indexes for word-break table changes

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.

PROUTIL offers options to optimize the performance of rebuilding indexes and the ability to limit the number of indexes to be rebuilt. However, the minimum syntax of PROUTIL with the IDXBUILD qualifier to rebuild all indexes is:

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

The syntax of PROUTIL with the IDXFIX qualifier is:

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

For the complete syntax of the PROUTIL utility, see *Manage the OpenEdge Database*.

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.

For details, see the following topics:

- Guidelines and methodology
- Input and output
- Data-processing issues
- Sorting data

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 OpenEdge 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.

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.

Processing by characters

To keep your application code flexible so 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 two bytes always equals two characters. By default, ABL functions process character data as whole characters, not as bytes. Make sure 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 ABL code processes data by characters, not bytes. As a result, it calculates the storage required incorrectly:

```
DEFINE VARIABLE iFloppySize AS INTEGER NO-UNDO INITIAL 1457664.
DEFINE VARIABLE iTotal      AS INTEGER NO-UNDO.

OUTPUT TO namelist.txt.
FOR EACH Customer NO-LOCK WHILE iTotal < iFloppySize:
  EXPORT Customer.Name.
  /* Calculate file size in bytes by adding name length, 2 quotes, carriage
     return and linefeed. */
  ASSIGN iTotal = iTotal + LENGTH(Customer.Name) + 4.
END.

OUTPUT TO terminal.
DISPLAY "NAMELIST.TXT is " iTotal "bytes".
IF iTotal >= iFloppySize 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. Since by default, however, 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(Customer.Name,"RAW")` instead of `LENGTH(Customer.Name)`. Using the `LENGTH` statement with *`type`* set to `RAW` tells OpenEdge 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 it maps to the hexadecimal value 82 in the IBM850 code page.

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.

Coding with translation in mind

OpenEdge includes, as an option, the Translation Management System, a product that supports the translation of OpenEdge 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 = TRUE.
```

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 OpenEdge AppBuilder generates also follows this convention. That is, AppBuilder puts `:U` after each string not to be translated.

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 Preparing the User Interface on page 73 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.

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.

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 Deployment and Configuration on page 143 for information on customizing OpenEdge 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 Using Multi-byte Code Pages on page 105

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 OpenEdge 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 `progress.ini`.

OpenEdge 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 *Manage ABL Applications* and *Configure OpenEdge*.

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-inch paper might lose a column on A4 paper.

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.

Numeric formats

Numeric data has a cultural component that any international OpenEdge application must address. Numeric separators (decimal and thousands separators) differ across locales. The following table shows how various countries use periods (.), commas (,), and spaces in numeric values.

Table 5: 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, OpenEdge 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, OpenEdge 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 NO-UNDO.
DEFINE VARIABLE price1 AS DECIMAL NO-UNDO.
...
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.

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 also can use the same data for all locales; all you have to do is update a conversion routine to reflect a new display format.

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.

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 16-02-2014, 2/16/2014, or other formats. 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 five is Friday), but might not be appreciated in New York where day five 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, OpenEdge 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 an OpenEdge session. The internal storage format remains the same. The `-d` startup parameter is only a client startup parameter
- The `DATE-FORMAT SESSION` handle attribute allows you to specify the format for displaying dates during the current OpenEdge session
- The Century (`-yy`) startup parameter allows you to specify the current start date for the OpenEdge two-digit year range of 100 years. See its reference entry in *Startup Command and 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 OpenEdge Resources on page 151 and *Startup Command and Parameter Reference*. For more information on the functions, see *ABL Reference.*

The following is an example of how to handle dates:

snhndl.p

```
DEFINE VARIABLE mynum AS DECIMAL NO-UNDO.

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 h-snhndl.p procedure illustrates the read and write capability of the DATE-FORMAT and NUMERIC-FORMAT SESSION handle attributes. Note OpenEdge 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.

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, 14.42.10, or 14h42:10. To accommodate these differences, you can use techniques similar to those used for dates. OpenEdge 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.

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 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.

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.

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 OpenEdge sorts and compares character data by using OpenEdge collations and International Components for Unicode (ICU) collations. A collation assigns values to characters; OpenEdge 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 ABL CAPS and LC functions perform these case conversions, OpenEdge uses a case table to match a character to its uppercase or lowercase equivalent.

Using collation tables

Each database has its own collation tables OpenEdge uses for index operations. ABL 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 OpenEdge 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, OpenEdge 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, OpenEdge goes to cell 233 in the table. After OpenEdge locates the appropriate cell, it reads the value in the cell to find the character's sort weight. The *sort weight* tells OpenEdge 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, OpenEdge 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 OpenEdge decides which collation table to use when performing comparisons or sorting:

- If you specify collation tables with the Collation Table (`-cpcoll`) startup parameter, OpenEdge uses those collation tables.
- If you name databases in the startup or connection command and do not use `-cpcoll`, OpenEdge uses the collation table defined for the first database. If you connect to an additional database, OpenEdge 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`, OpenEdge 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 `OpenEdge/convmap.cp` file, which is a binary file that contains tables for managing characters created by compiling the `OpenEdge/prolang/convmap/convmap.dat` file. If `convmap.dat` does not have a collation table you need, you can create your own.

ABL Comparisons

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

OpenEdge supplies collation tables for databases for many languages and locales. These are located in the `OpenEdge/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 collations and collation tables, see Understanding Character Processing Tables on page 41. For information on using ICU collations as database collations, see Using Databases on page 85,

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." In Canada, however, the character "é" retains the accent and becomes "É" when capitalized.

OpenEdge 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(Customer.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(Customer.Name) MATCHES "RENE":
```

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

OpenEdge has a default case table for every code page that it supports. The case tables are in the `OpenEdge/prolang/convmap.dat` file. For more information on the `convmap.dat` file, see Understanding Code Pages on page 27. 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. OpenEdge 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 an OpenEdge client session or server.

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

- If you specify a case table with the `-cpcase` startup parameter, OpenEdge uses that case table
- If you do not use `-cpcase`, OpenEdge 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 OpenEdge 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 `OpenEdge/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 Understanding Character Processing Tables on page 41.

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.

This means designing 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.
- Capable of working 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.

For details, see the following topics:

- Screen layout and composition
- Graphics and icons
- Designing with the AppBuilder
- Designing to allow for translation

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.

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.

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.

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.

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: telephone

French: 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. OpenEdge 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 on page 75
- Address formats on page 75
- Calendar, date, and time formats on page 76
- Colors and sounds on page 76
- Numeric formats on page 76
- Currency formats on page 76
- Field labels and field sizes on page 77
- Financial rules on page 77
- Names and titles on page 77
- Phone number formats on page 77

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, and so on, 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. For example, the same date could be represented as 16-02-2014 or 2/16/2014, and the same time could be represented as 2:42:10 PM or14.42.10. Punctuation and capitalization in the date and time format also vary. A slash (/) might separate the numbers in a date format (as in 2010/9/18) or a period (.) as in (2010.5.4). 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.

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 0.5 as the determining value. For example, round numbers in the range 2.1-2.4 down to 2.0, and numbers in the range 2.5-2.9 up to 3.0.
- Round numbers up or down using 0.25 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 previous examples is an acceptable phone number format. You should always 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. Four of the many language issues follow:

- **Direction in which 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 (IMEs)** — 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.

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.

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.

Images to avoid in graphics and icons

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

- Images of animals on page 79
- Images based on puns or word play on page 79
- Images of body parts or gestures on page 79
- Images biased towards a particular culture or region on page 79
- References to sports on page 79
- Symbols related to religion or culture on page 79

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.

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 on page 80
- Localizing SmartObject instances on page 80
- Using multiple layouts on page 82

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.

The following figure 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. The figure shows a SmartObject with an Irish address.

Figure 7: SmartObject labels

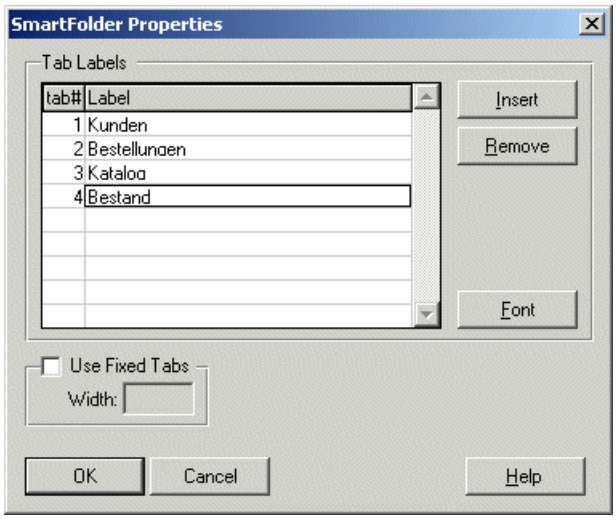


Localizing SmartObject instances

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.

The following figure illustrates how you can change SmartFolder tab labels in the **SmartFolder Properties** dialog box to create a customized running instance of a SmartFolder.

Figure 8: SmartFolder Properties dialog box



The following figure shows a customized running instance of a SmartFolder.

Figure 9: Localized SmartFolder



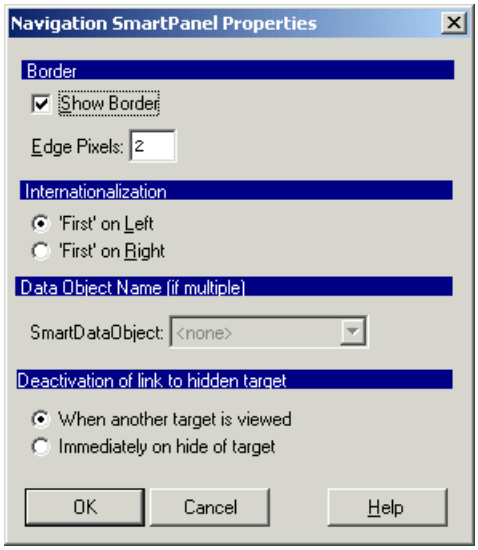
You can localize a SmartPanel with the **Navigation SmartPanel Properties** dialog box. For example, a graphical Navigation SmartPanel has buttons that represent the functions; First, Next, Previous, and Last. The following figure shows a graphical Navigation SmartPanel.

Figure 10: 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. The following figure shows the **Navigation SmartPanel Properties** dialog box.

Figure 11: Navigation SmartPanel Properties dialog box



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.

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 the AppBuilder Character Run window uses the correct DOS code page:

1. Create a parameter file (`.pf` file) for the character client. Make sure `-cpinternal` and `-cpstream` are set to the appropriate code page for DOS.
2. Specify the pathname of the OpenEdge character client startup file with the `PROSTARTUP` environment variable in the `[WinChar Startup]` section of the `progress.ini` file or Registry, as shown:

```
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, OpenEdge error messages might be displayed incorrectly if the character session uses a multi-byte `PROMSGS` file.

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 on page 83
- Message text on page 84
- Layout on page 84

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

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 on page 83
- Typefaces and point sizes on page 84
- Word order on page 84

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.

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. For more information on the `SUBSTITUTE` function, see *ABL Reference*.

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.

Some messages have multiple translations. 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.

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 it does not wrap to a second line when translated.

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 those rules and techniques.

For details, see the following topics:

- OpenEdge 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

OpenEdge 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:

- ABL reserved words
- Letters with diacritical marks

- The following characters:

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

For more information on OpenEdge database name restrictions, see *Manage the OpenEdge Database*. For more information on ABL reserved words, see *ABL Reference*.

Empty databases

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

Empty databases supplied by OpenEdge

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

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

Creating an empty database in a particular code page

This section describes a technique for producing an empty database in a particular code page, which is included here for completeness.

To create an empty database with a code page:

1. Create an empty database, using the OpenEdge Data Dictionary utility.
2. Within the OpenEdge distribution, select the `OpenEdge/prolang` directory for the target locale.

   For example, the `OpenEdge/prolang/cze` directory contains files used for Czech databases, while the `OpenEdge/prolang/utf` directory contains files used for Unicode databases.

3. Within the `OpenEdge/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 `OpenEdge/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

Note: The `OpenEdge/prolang/utf` directory contains the collation data definition files for the International Components for Unicode (ICU) collations (such as `ICU-cs.df` used for Czech databases).

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`, which contains the UTF-8 BASIC collation.

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

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

   For the complete syntax of the `PROUTIL` utility, see *Manage the OpenEdge Database*.

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

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.

Now 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.

New characters

Code pages 1252 and ISO8859-15, listed in the following table, contain characters not in ISO8859-1.

Table 6: 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

Character	Position in 1252	Position in ISO8859-15
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

The following table 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 7: Characters to check for before converting to 1252

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

The following table 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 8: 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

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.

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 to which you are converting.

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 *Manage the OpenEdge Database*.

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"
```

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 seen in Table .

Another reason 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.

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 Understanding Code Pages on page 27. For the complete syntax of the PROUTIL utility, see *Manage the OpenEdge Database*.

Loading table dump files

When a table dump file loads, it is essential that OpenEdge interprets the table dump file's data correctly—that is, according to the code page the data was encoded in. If OpenEdge 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.

Code-page trailers

Some table dump files contain code-page trailers, which tell OpenEdge the code page of the table dump file's data. When you load such a table dump file, OpenEdge reads the code-page trailer, determines the code page of the file's data, and interprets the data correctly. In the current version of OpenEdge, when you dump a database using the Data Dictionary utility, the resulting table dump file contains a code-page trailer.

The following figure shows a typical code-page trailer with the code page name "IBM850."

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
```

Other table dump files do not contain code-page trailers. When you load such a table dump file, you must tell OpenEdge 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 Understanding Code Pages on page 27.

Techniques for loading table dump files

There are several techniques for loading a table dump file, described below.

Using an ABL program

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

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

Note: An ABL program like 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 ABL 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.

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:

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

For the complete syntax of the `PROUTIL` utility, see *Manage the OpenEdge Database*.

Specifying -cpinternal and -cpstream with database utilities

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

7

Using SQL

OpenEdge provides several programming languages. One of these is SQL, the focus of this chapter.

Note: For a complete description of the OpenEdge SQL language, see *Develop SQL for OpenEdge*.

For details, see the following topics:

- Starting database servers
- Using SQL database servers
- Code-page conversion in SQL applications
- Using SQL clients
- Using SQL utilities
- Using the SQL language

Starting database servers

You can start a database server in two ways:

- By invoking `PROSERVE` from a command line or from an OpenEdge client
- By using OpenEdge Explorer or OpenEdge Management , a graphical application

When an OpenEdge client invokes PROSERVE, PROSERVE starts a database server on behalf of the client. For a ABL client, PROSERVE starts a ABL database server. For an SQL client, PROSERVE starts an SQL database server.

If a client invokes PROSERVE with startup parameters, such as -cpinternal, -cpstream, and -cpcoll, PROSERVE passes these values to the ABL or SQL database server. The ABL database server uses these values. The SQL database server, however, ignores some of these values. Specifically, the SQL 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 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 OpenEdge Explorer or OpenEdge Management you can specify a code page and a collation by using the Explorer's administrative features.

The following table lists localizable startup parameters and settings for SQL and ABL database servers.

Table 9: 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	NA	IBM850
-cplog	IBM850	Log character set	ISO8859-1
-cpcase	BASIC	Case table	BASIC
-cpcoll	BASIC	Collation table	BASIC
-convmap	OpenEdge/convmap.cp	Conversion map	OpenEdge/convmap.cp

OpenEdge Explorer and OpenEdge Management also let 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:

1. From OpenEdge Explorer or OpenEdge Management, view a log file.
2. From the **Action** menu, select **Font...**. Alternatively, click on the toolbar Font icon. The standard Windows **Font Selection** dialog box appears.
3. In the **Font Selection** dialog box, choose a font, then click **OK**. OpenEdge Explorer or OpenEdge Management displays the log file using the font you selected.

Using SQL database servers

When you use SQL database servers, the following applies:

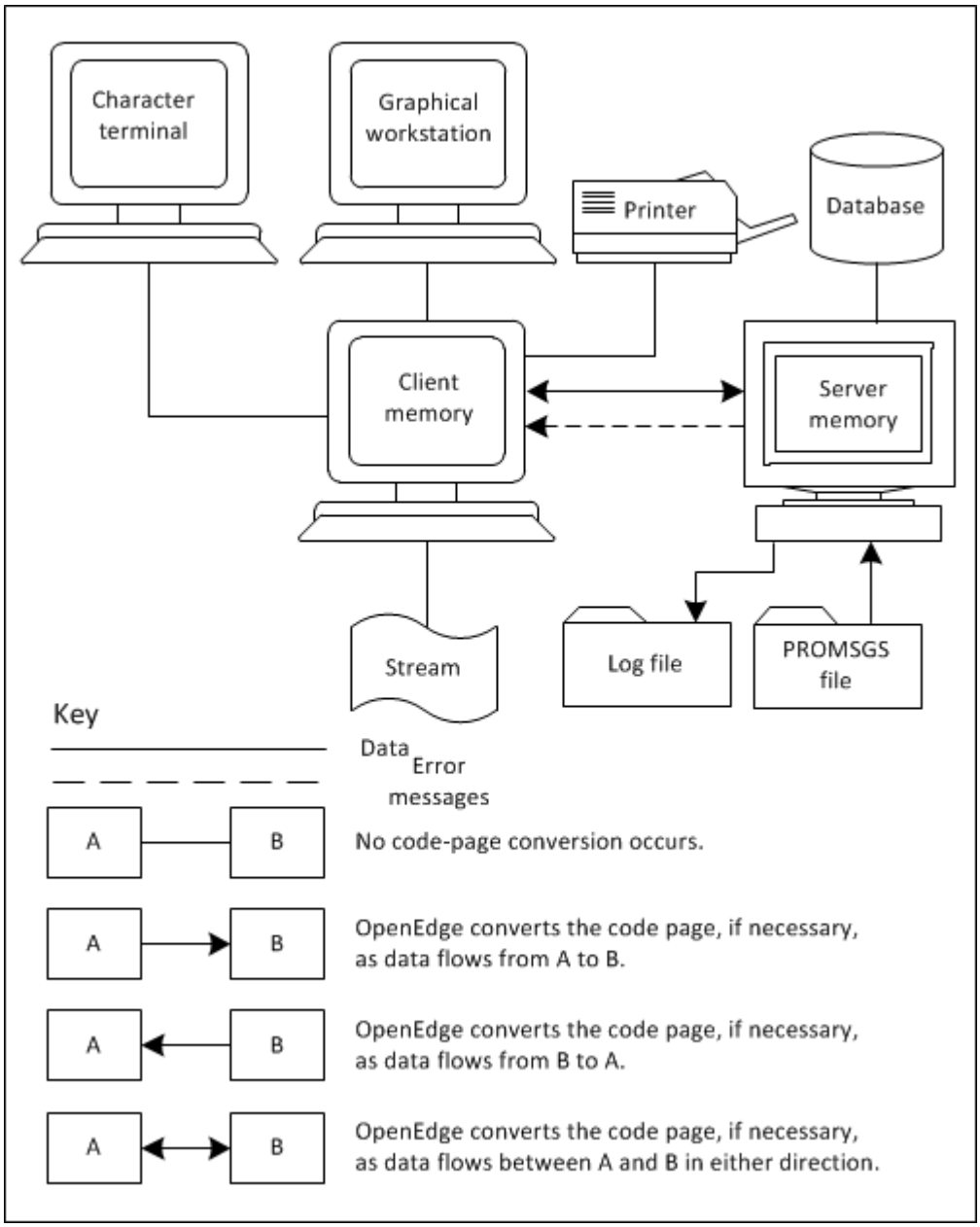
- The SQL database server is multi-threaded. This means each server process can handle multiple client connections.
- The SQL server uses the current OpenEdge `convmap.cp` file. This is the one the `PROCONV` environment variable, if set, points to; otherwise, this is the one in the `OpenEdge` directory.
- The SQL server uses the code page of the connected database as its internal code page. Similarly, the SQL 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 database server uses the same `PROMSGS` file as `PROSERVE`. This file is the one specified by the `PROMSGS` environment variable, if it is set. Otherwise, this file is the `promsgs` file in the `OpenEdge` directory.

Code-page conversion in SQL applications

In SQL applications, when OpenEdge performs code-page conversions, these generally occur on the server. In ABL applications, by contrast, these generally occur on the client.

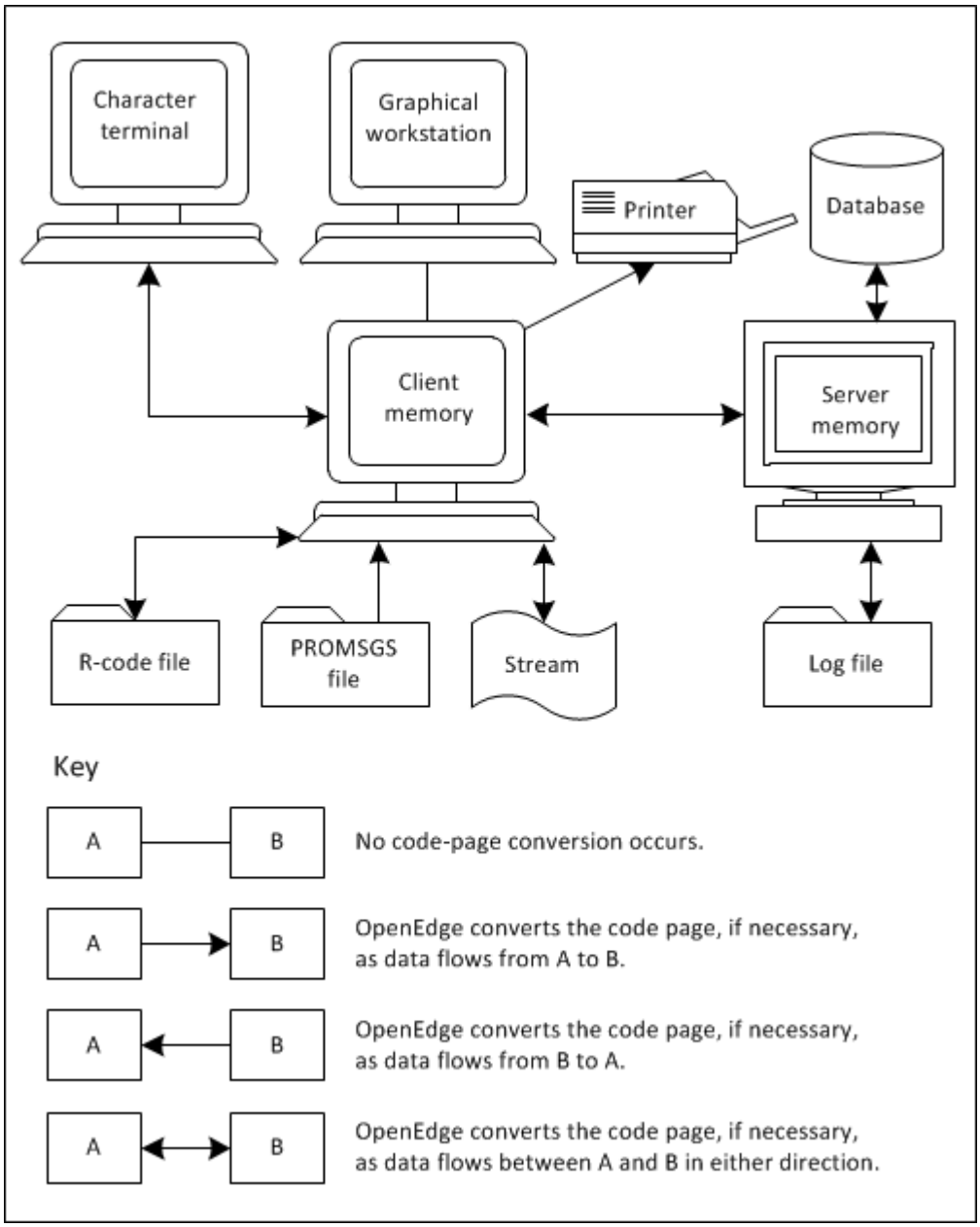
The following figure shows a typical SQL application and the code-page conversions performed by OpenEdge.

Figure 12: Code-page conversion in an SQL application



Compare this figure with the following figure, which shows a typical ABL application and the code-page conversions performed by OpenEdge.

Figure 13: Code-page conversion in an ABL application



Using SQL clients

SQL databases can have the following types of clients:

- Java/Java Database Connectivity (Java/JDBC)
- Open Database Connectivity (ODBC)

Java/JDBC clients

Java/JDBC clients, in source code form, consist of statements in the Java language. Java/JDBC clients communicate with SQL 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 OpenEdge SQL Explorer. For more information on building Java/JDBC clients, see *Develop SQL for OpenEdge*.

ODBC clients

ODBC clients access OpenEdge SQL 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 *Develop SQL for OpenEdge*.

Characteristics of OpenEdge SQL clients

OpenEdge SQL clients have the following characteristics:

- 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 Using Unicode on page 127.

For ODBC clients, the internal code page is the internal code page is that of the client's default locale.

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 the code page of the client's default locale.

Using SQL utilities

OpenEdge provides several utilities for working with SQL databases, including `SQLDUMP`, `SQLLOAD`, and `SQLSCHEMA`.

SQLDUMP

SQLDUMP lets you dump SQL databases. The syntax for SQLDUMP (for UNIX and Windows) is:

```
sqldump -u username -a password -t table-name [,table-name . . . ] database-url [-C
 charset] [-F { comma | quote } ] [-f command-filename] [-n error-check]
```

username, *password*

User authentication.

table-name [,*table-name* . . .]

One or more tables to dump. Multiple tables can be specified, in a comma-separated list. Specify a *table* of % to dump all tables. The -t option can be specified only if the -f option is not.

database-url

The database to which you want to connect, which has the following form:

```
progress:T:host-name:service-name:database-name
```

charset

The name of the character set of the tables to be dumped.

{ comma | quote }

If comma is specified, dump in comma-delimited format. If quote is specified, dump in quote-delimited format. The default is quote.

command-filename

Specifies the file where SQLDUMP reads commands on how and what to dump. The -f option can be specified only if the -t option is not.

error-check

Check for syntax errors without dumping any rows.

Note:

- The SQLDUMP utility does not support dumping a table containing a CLOB or BLOB column. A request to dump a table with a CLOB or BLOB column will result in an error message, and that table will be skipped.
- To dump a table containing a CLOB or BLOB column, use the DB binary dump/ PROUTIL utility. For detailed information, see *PROUTIL DUMP qualifier* topic of the *OpenEdge Data Management: Database Administration* guide.

SQLLOAD

SQLLOAD lets you load SQL databases. The syntax for SQLLOAD (for UNIX and Windows) is:

```
sqlload -u username -a password -t table-name [,table-name . . . ] database-url [-b
 bad-row-filename] [-C charset] [-e max-errors] [-F { comma | quote } ] [-f
command-filename] [-l log-filename] [-n error-check]
```

username, *password*

User authentication.

table-name [,*table-name* . . .]

One or more tables to load . Multiple tables can be specified, in a comma-separated list. Specify a *table* of % to load all tables. The -t option can be specified only if the -f option is not.

database-url

The database to which you want to connect, which has the following form:

```
progress:T:host-name:service-name:database-name
```

bad-row-filename

The file where SQLLOAD writes rows that were not loaded. The default is screen.

charset

The name of the character set of the tables to be loaded.

maxerrors

The maximum number of errors allowed before processing is terminated. The default is 50.

{ comma | quote }

If comma is specified, load in comma-delimited format. If quote is specified, load in quote-delimited format. The default is quote.

command-filename

The file where SQLLOAD reads commands on how and what to load. The -f option can be specified only if the -t option is not.

log-filename

The file where SQLLOAD writes errors and statistics. The default is screen.

error-check

Check for syntax errors without loading any rows.

SQLSCHEMA

SQLSCHEMA is a command-line utility that writes selected components of an SQL database schema to an output file using the UTF-8 code page. You might then use SQL Explorer to load the schema components into an SQL database The syntax for SQLSCHEMA (for UNIX and Windows) is:

```
sqlschema -u username -a password -t table-name [,table-name . . . ] database-url [-g
 table-list . . . ] [-o output-filename] [-p procedure-list] [-s synonym-list] [-T
 trigger-list]
```

username, *password*

User authentication.

table-name **[**,*table-name* **. . .]**

Table schemas to dump. Multiple tables can be specified, in a comma-separated list. Specify a *table* of % to load all tables. The syntax supports the % (percent) and _ (underscore) special characters and the \ (backslash) escape character.

database-url

The database to which you want to connect, which has the following form:

```
progress:T:host-name:service-name:database-name
```

table-list

Comma-separated list of one or more grant privileges to dump as grant statements. The syntax supports the % (percent) and _ (underscore) special characters and the \ (backslash) escape character.

output-filename

The file to which schemas are dumped. The default is screen.

procedure-list

Comma-separated list of one or more procedures for which to capture definitions. The syntax supports the % (percent) and _ (underscore) special characters and the \ (backslash) escape character.

synonym-list

Comma-separated list of one or more synonyms to dump as `create synonym` statements. The syntax supports the `%` (percent) and _ (underscore) special characters and the `\` (backslash) escape character.

trigger-list

Comma-separated list of triggers for which to capture definitions. The syntax supports the `%` (percent) and _ (underscore) special characters and the `\` (backslash) escape character.

Using the SQL language

When you use the SQL language in applications that you intend to localize to multiple locales, the following considerations 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 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 OpenEdge SQL, see OpenEdge Resources on page 151. For a complete description of the OpenEdge SQL language, see *Develop SQL for OpenEdge*.

Unit of length when working with character strings

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

Consider the syntax of the OpenEdge SQL `LEFT` function:

Table 10:

```
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□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□B
```

Maximum bytes required for CHAR and VARCHAR

For SQL, the bytes required for `CHAR` or `VARCHAR` fields 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.

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, Korean, and also touches on Unicode. For more information on Unicode, see Using Unicode on page 127.

For details, see the following topics:

- Definitions of key terms
- OpenEdge support
- Input
- Output
- Inside the multi-byte application
- Issues specific to multi-byte code pages
- Guidelines for using multi-byte characters

Definitions of key terms

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

- Terms for code pages on page 106

- Terms for characters on page 106
- Terms for bytes on page 106

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. For example, UTF-8 is a 4-byte code page
- A *multi-byte* code page is a code page that is not single byte.

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.

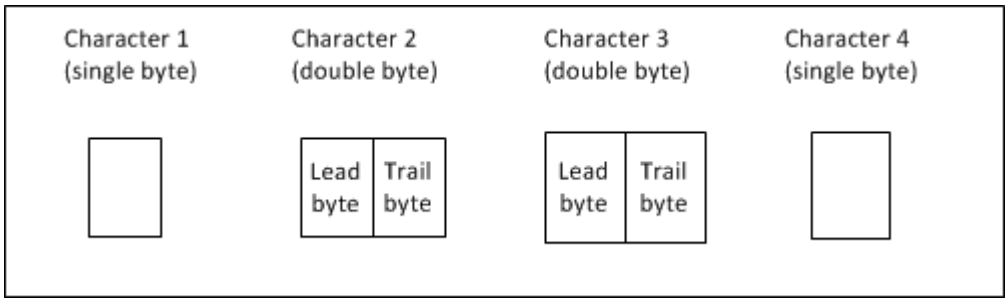
Terms for bytes

The following terms describe bytes:

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

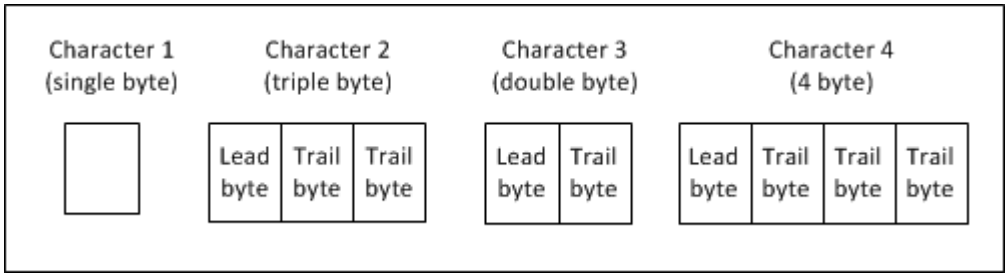
The following figure shows four characters from a double-byte code page.

Figure 14: Four characters from a double-byte code page



The following figure shows three characters from a UTF-8 code page. Note that Character 4 is a Unicode supplementary character.

Figure 15: Three characters from a UTF-8 code page



OpenEdge support

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

Multi-byte characters are supported by the majority of OpenEdge products. The following table lists each OpenEdge product and the multi-byte support it provides.

Table 11: Support for multi-byte characters

Product	Multi-byte support
AppBuilder	The AppBuilder—except for the Character Run window—supports double-byte and UTF-8 characters.Widgets created with the AppBuilder can have double-byte or UTF-8 characters in labels and in text.The Character Run window does not support double-byte or UTF-8 characters.
AppServer	The AppServer supports double-byte and UTF-8 characters.
Application Compiler	The Application Compiler supports double-byte and UTF-8 characters. That is, the ABL source code that the Application Compiler compiles can contain double-byte and UTF-8 characters, and the resulting r-code supports double-byte and UTF-8 characters.
Progress Developer Studio for OpenEdge	The Progress Developer Studio for OpenEdge supports double-byte and UTF-8 characters.

Product	Multi-byte support
Clients (graphical, character, and batch)	The graphical client supports double-byte characters and UTF-8 characters.The UNIX character client supports double-byte characters but not UTF-8 characters.The Windows character client does not support multi-byte characters.The batch client (started with the command `prowin32 -b`) supports multi-byte characters.
Data Dictionary	The OpenEdge Data Dictionary supports multi-byte characters.
DataServers	The DataServer for Oracle supports double-byte and UTF-8 characters if the ORACLE DBMS is set up for Native Language Support (NLS). For more information on NLS, see the Oracle documentation.
HLC Applications	An HLC program can have double-byte and triple-byte characters.Since the AVM 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, the AVM searches each character string for the NULL terminator. For each multi-byte character in the string, the AVM 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, the AVM misses it and does not detect the end of the string correctly.
Procedure Editor	On Windows, the Procedure Editor supports double-byte and UTF-8 characters; on UNIX, only double-byte characters.To write a multi-byte ABL 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 UTF-8 characters.
SQL	SQL supports double-byte and UTF-8 characters.
Translation Management System	The Translation Management System can manage translations to and from languages that use double-byte and UTF-8 code pages.
WebClient™	WebClient supports double-byte characters and UTF-8 characters.
WebSpeed	WebSpeed fully supports multi-byte characters.

Input

This section describes how to get multi-byte characters into your application. It includes the following subsections:

- Using the keyboard and mouse on page 109
- Using ABL on page 110
- Automatic input validation on page 112

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, OpenEdge does not provide its own input methods or IMEs. Rather, OpenEdge supports the input methods and IMEs that Microsoft supplies and those that fully support the Microsoft standard.

Data flow in single-byte applications

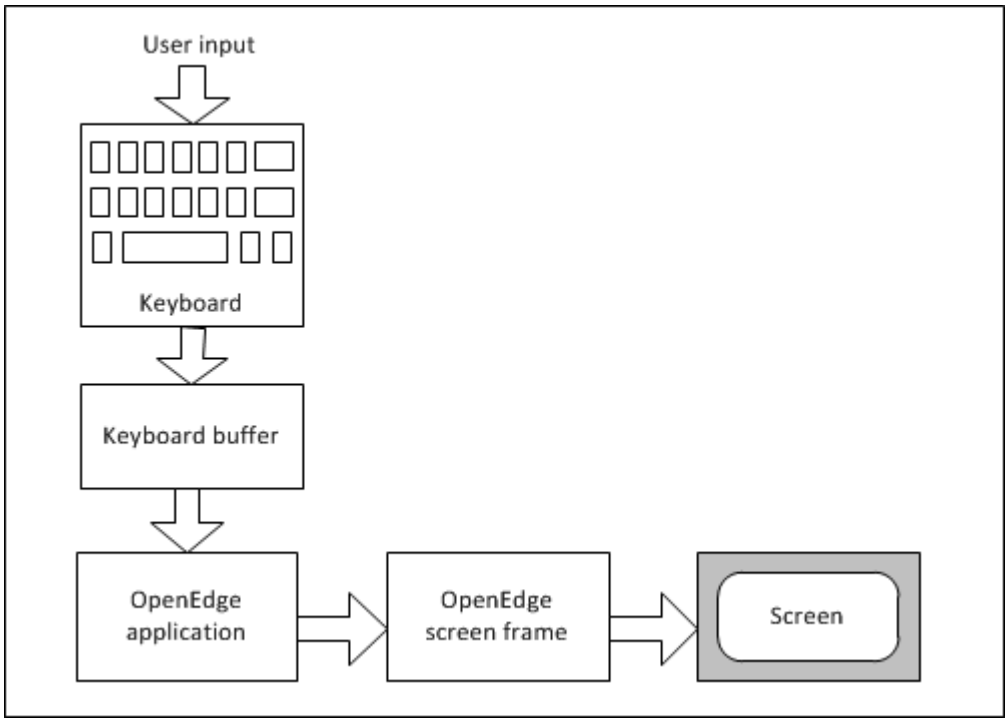
Using an IME affects the flow of input through OpenEdge applications. In single-byte applications, which do not need IMEs, input data travels as follows:

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

Once the input data reaches the keyboard buffer, the programmer can query it using the ABL `LASTKEY` function and `READKEY` statement.

The following figure illustrates the flow of data through an application that does not use an input method editor.

Figure 16: Data flow through an application not using an input method editor



Data flow in double-byte applications

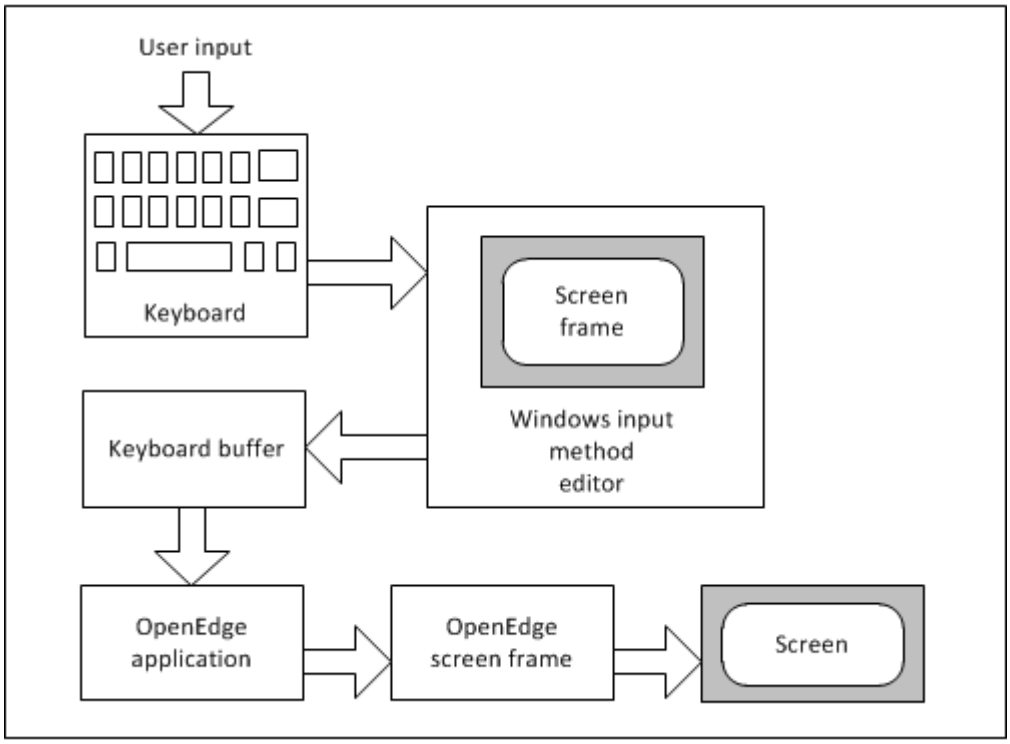
In double-byte applications, which use IMEs, input data travels as follows:

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

As before, once input data reaches the keyboard buffer, the programmer can query it using the ABL `LASTKEY` function and `READKEY` statement.

The following figure illustrates the flow of data through an application that uses an input method editor.

Figure 17: Data flow through an application using an input method editor



Using ABL

Besides using the keyboard and possibly the mouse to enter multi-byte characters, you can also use ABL.

Using ABL to simulate inputting double-byte characters

To input a double-byte character using ABL, assemble the character using the CHR function. This is the syntax:

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

The following example creates and displays the double-byte character shown below:



It is from the KSC5601 code page, which supports Korean:

```
DEFINE VARIABLE cDoubleByteChar AS CHARACTER NO-UNDO.
cDoubleByteChar = CHR((236 * 256) + 237).
DISPLAY cDoubleByteChar 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 the internal code page (-cpinternal) is set to KSC5601.

Using ABL to simulate key presses and mouse clicks

Besides using ABL to simulate the input of characters, including multi-byte characters, you can also use ABL to simulate key presses and mouse clicks in a multi-byte environment.

The following table describes the ABL elements involved and the multi-byte support provided by each.

Table 12: ABL elements that simulate key presses and mouse clicks

ABL 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.

Automatic input validation

When you input multi-byte characters, OpenEdge automatically validates the input under certain conditions. Validation of multi-byte characters means OpenEdge 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 a multi-byte character is valid. Though its lead byte has an appropriate lead-byte value and each of its trail bytes has an appropriate trail-byte value, the combination of values still might be invalid for the character's code page.

The following table describes when validation does and does not occur.

Table 13: Automatic validation of multi-byte input

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

Output

OpenEdge allows you to 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 Deployment and Configuration on page 143. The rest of this section discusses the `FORMAT` phrase and printing.

FORMAT phrase

The OpenEdge Data Dictionary Format field and the ABL `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. One-column characters exist in single-byte, double-byte, or UTF-8 code pages.
- Four, two-column characters. Two-column characters exist in double-byte and UTF-8 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, OpenEdge ensures that the display or print field contains enough columns to accommodate the entire multi-byte character. If there are not enough columns, OpenEdge does not display any of the multi-byte character and pads the columns of the display field or print field with spaces.

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

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

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

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

```
"日日日日"
```

OpenEdge displays the first three double-byte characters in columns 1 through 6 of the field and a space in column 7:

```
"日日日  "
```

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

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

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

You want 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, the AVM:

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 only single-byte formatting characters

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 "□xx" /*invalid format string */
```

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 on page 114
- Printing from applications with a character interface on page 114
- Testing printers on page 115

Printing from Windows

To print from a multi-byte OpenEdge 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 *Manage ABL Applications*.

Printing from applications with a character interface

To print from a multi-byte OpenEdge application with a character interface, make sure 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 through 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 OpenEdge. 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.

Character-client color limit

When you are using the ABL character client with a double-byte code page, the number of colors is limited to 31.

To deal with the 31-color limit:

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 *Manage ABL Applications*.

Inside the multi-byte application

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

This section discusses the following topics:

- Distinguishing characters, bytes, and columns on page 115
- Techniques for working with multi-byte characters on page 116
- ABL support for processing multi-byte characters on page 120

Distinguishing characters, bytes, and columns

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, examine the following table, which shows two characters, one single-byte and one double-byte, and the byte and column count of each.

Table 14: Byte count and column count

Character	Number of bytes	Number of columns
A	1	1
中	2	2

Techniques for working with multi-byte characters

The following techniques might save you time and trouble:

- Choosing the appropriate unit of measure on page 116
- Testing character strings for multi-byte characters on page 117
- Testing for a lead-byte value on page 118

Choosing the appropriate unit of measure

Several ABL elements, including the `LENGTH` function, `OVERLAY` statement, `SUBSTRING` function, and `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 cCharOver AS CHARACTER FORMAT "X(8)" NO-UNDO.

cCharOver = "abc中efg".

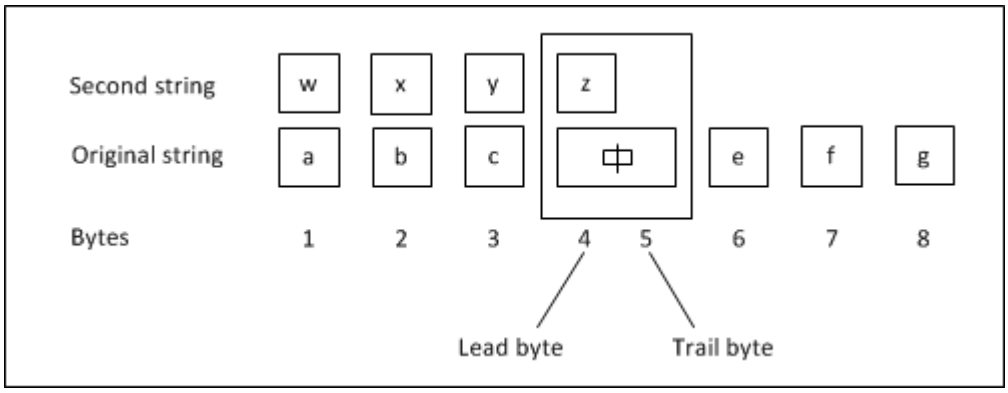
OVERLAY(cCharOver), 1, 4, "RAW") = "wxyz". /* RAW is wrong */

DISPLAY cCharOver 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, single-byte characters 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.

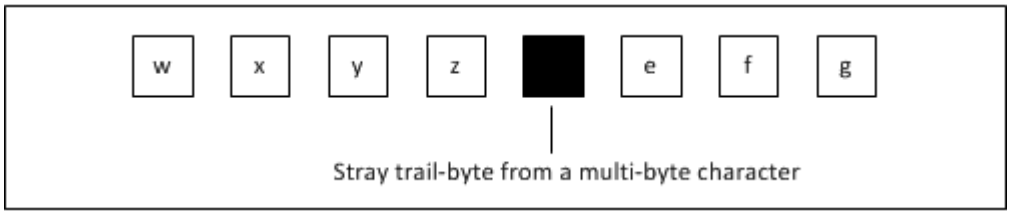
The following figure shows how the z in the second string overlays the lead byte of the double-byte character in the original string.

Figure 18: A single-byte character overlaying a lead byte



All that remains of the multi-byte character is the trail-byte, as shown in the following figure.

Figure 19: Result of a single-byte character overlaying a lead byte



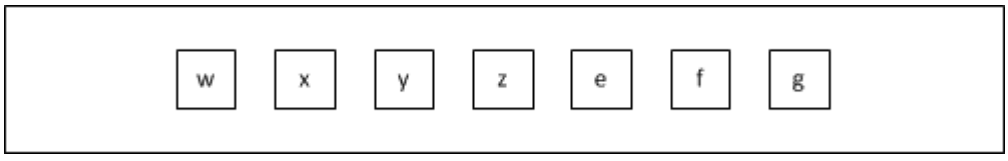
To fix this error, change the unit of measure to `CHARACTER`, as shown:

```
DEFINE VARIABLE cCharOver AS CHARACTER FORMAT "X(8)" NO-UNDO.

cCharOver = "abc中efg".
OVERLAY(cCharOver), 1, 4, "CHARACTER") = "wxyz".
/* CHARACTER is correct */
DISPLAY cCharOver WITH 1 COLUMN.
```

The corrected program produces the string shown in the following figure.

Figure 20: 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:

```
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. The 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, OpenEdge validates the type specification at compile time. If you specify the *type* as a variable expression, OpenEdge 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; otherwise, 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 "中" NO-UNDO.
IF LENGTH(mychar,"CHARACTER") = LENGTH(mychar,"RAW") THEN
  DISPLAY "No multi-byte characters 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" NO-UNDO.
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. The following table lists the lead-byte and trail-byte values for the multi-byte code pages OpenEdge supports.

Table 15: 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 126161 through 254
CP949	Korean	129 through 254	65 through 9097 through 122129 through 254
CP950	Traditional Chinese	129 through 254	64 through 126128 through 254
CP936	Simplified Chinese	129 through 254	64 through 126128 through 254
CP1361	Korean	132 through 211216 through 222224 through 249	65 through 127129 through 254
EUCJIS	Japanese	142164 through 254	161 through 254
GB2312	Simplified Chinese	161 through 254	161 through 254
GB18030[1]	Extended Chinese	-	-
KSC5601	Korean	161 through 254	161 through 254
SHIFT-JIS	Japanese	129 through 159224 through 252	64 through 126128 through 252
UTF-8	Unicode	193 through 239	128 through 191

You cannot always assume a byte with a lead-byte value is a lead byte, or 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 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 `YES` 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 `NO`. `IS-LEAD-BYTE` has the following 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.

[1] The GB18030 code page is a multi-byte code page, consisting of one-, two-, and four-byte characters, that extends the GB2312 code page and includes all characters defined in Unicode. Unlike most multi-byte code pages that OpenEdge supports, you cannot use the lead byte of multi-byte characters in the GB18030 code page to determine the character's length. OpenEdge uses the International Components for Unicode (ICU) library to convert characters between the GB18030 code page and Unicode within the OpenEdge GUI client.

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 `NO`, and the example displays `Lead: no`:

```
DEFINE VARIABLE lLead AS LOGICAL NO-UNDO.
lLead = IS-LEAD-BYTE("xy中").
DISPLAY lLead 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 `YES`, and the example displays `Lead: yes`:

```
DEFINE VARIABLE lLead AS LOGICAL NO-UNDO.
lLead = IS-LEAD-BYTE("中xy").
DISPLAY lLead WITH 1 COLUMN.
```

ABL support for processing multi-byte characters

ABL supports multi-byte characters in general and has many elements that support multi-byte characters in particular. For a list of these language elements, see OpenEdge Resources on page 151.

Issues specific to multi-byte code pages

Issues specific to multi-byte code pages are discussed in the following sections:

- OpenEdge support for multi-byte code pages on page 120
- Valid and invalid code-page conversions on page 122
- User-defined characters on page 123
- Collating multi-byte characters on page 124
- Default word-break behavior of characters in multi-byte code pages on page 125

OpenEdge support for multi-byte code pages

The following table lists the multi-byte code pages OpenEdge supports.

Table 16: Multi-byte code pages OpenEdge supports

Code page	Language
BIG-5	Traditional Chinese
CP949	Korean

Code page	Language
CP936	Simplified Chinese
CP950	Traditional Chinese
CP950-HKSCS	Traditional Chinese with Hong Kong supplementary character set
CP1361Note: This is not convertible to UTF-8 and not recommended for use.	Korean
EUCJIS	Japanese
GB2312	Simplified Chinese
GB18030	Extended Chinese (that is, an extension of GB2312 that includes all characters defined in Unicode)
KSC5601	Korean
SHIFT-JIS	Japanese
UTF-8	Multi-lingual
UTF-16, UTF-16BE, UTF-16LE, UTF-32, UTF-32BE, UTF-32LE	Multi-lingual; for input and output, not for `cpinternal`

About the Thai code page and collations

The Thai language is supported by code page TIS 620-2533. This is a single-byte code page for which some of the codepoints are combining characters. A single column could consist of up to three 620-2533 codepoints; however, it is not a multi-byte code page consisting of lead-byte and trail-byte combinations.

There are two files for Thai collation:

- **`OpenEdge/prolang/tai/thai-old.df`** — Contains the `"THAI"` collation for codepage TIS 620-2533. This collation assigns a single weight to each character for case-sensitive and case-insensitive sorting. Although adequate for many applications, this collation does not follow Thai conventions for prefix vowels or treatment of strings as collation units of multiple characters. This collation was provided in Progress versions prior to 9.1A and 8.3C, and is still provided for backward compatibility.
- **`OpenEdge/prolang/tai/thai.df`** — Contains the `"BASIC"` collation for TIS 620-2533. This collation takes prefix vowels into account when sorting. It also treats strings as containing collation units of multiple characters.

To check for further information on Thai code page issues, you can find the following `readme` file in your OpenEdge installation if you specified Thai language support during the install process: `OpenEdge\prolang\tai\readme`.

Additional GB18030 code page considerations

When working with the GB18030 code page, consider the following:

- The GB18030 code page is a multi-byte code page, consisting of one-, two-, and four-byte characters, that extends the GB2312 code page and includes all characters defined in Unicode. Unlike most multi-byte code pages that OpenEdge supports, you cannot use the lead byte of multi-byte characters in the GB18030 code page to determine the character's length. OpenEdge uses the International Components for Unicode (ICU) library to convert characters between the GB18030 code page and Unicode within the OpenEdge GUI client.
- OpenEdge supports the GB18030 code page, primarily, for file input and output. More specifically, you can use the GB18030 code page for the following:
  - In settings for the `-cpstream`, `-cplog`, and `-cpprint` startup parameters
  - With any ABL `CONVERT SOURCE/TARGET` phrase that applies to files, for example:

```
INPUT FROM gb.txt CONVERT SOURCE "GB18030".
COPY-LOB FROM FILE "gb.txt" TO myunicodeclob
  CONVERT SOURCE CODEPAGE "GB18030".
```

  - With the `ASC`, `CHR`, and `CODEPAGE-CONVERT` functions (but if used as the target code page of `CHR` or `CODEPAGE-CONVERT`, you cannot use the resulting character or character string in ABL)
  - With file input and output in the SQL products
- You cannot use the GB18030 code page in the following instances:
  - In other ABL statements (not listed above)
  - As a CLOB or LONGCHAR code page
  - In settings for the `-cpinternal`, `-cpterm`, `-cprcodein`, or `-cprcodeout` startup parameters
  - As a database code page
  - With the `PROUTIL` *`db-name`* `-C CONVCHAR CONVERT` command (although it can be used for `CONVCHAR ANALYZE`)

Valid and invalid code-page conversions

Sometimes you can convert data from one code page to another and sometimes you cannot. Understanding Code Pages on page 27 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 OpenEdge is converting from UTF-8 and encounters a character that does not exist in the target code page, OpenEdge 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 from double byte to 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 the following table are valid.

Table 17: Valid double-byte to double-byte code-page conversions

Conversion	Comment
SHIFT-JIS to EUCJISEUCJIS 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

You cannot convert from CP949 to KSC5601, both code pages for Korean, because CP949 contains many characters that KSC5601 does not.

User-defined characters

OpenEdge 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, OpenEdge 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 an OpenEdge application has a tier (client, server, and so on) 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, remember to:

- Go into the `Japanese.datCONVMAP` file and, in the lead-byte attribute table, set the value of each byte in the range 240 through 252 to 1, to indicate a lead byte. You might only have to set some of them, depending on how many user-defined characters you want.

  The `Japanese.datCONVMAP` file resides in the `OpenEdge/prolang/convmap` directory. For more information on `CONVMAP` files, see Understanding Character Processing Tables on page 41.

- 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.

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 characters 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 OpenEdge provides for the double-byte Asian languages (Chinese, Japanese, and Korean) sorts all single-byte characters before all double-byte characters. The following table shows how OpenEdge sorts Japanese characters.

Table 18: Japanese collation order by character type

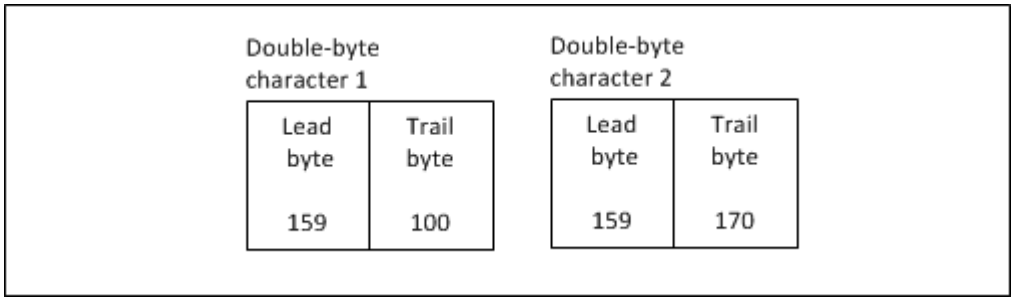
Character type	Range of values
Single byte (ASCII)	0-127
Single byte (half-width Katakana)	160-223
Lead byte (range 1)	129-159
Lead byte (range 2)	224-239
User-defined (Gaiji)	240-252

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 `OpenEdge/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, as illustrated in the following figure.

Figure 21: Sorting double-byte characters



Default word-break behavior of characters in multi-byte code pages

The following table describes the default word-break behavior of characters in multi-byte code pages. The table assumes word-break tables are Version 9 Type 3. For more information on word-break tables, see Word-break tables on page 53.

Note: The default word-break behavior can be changed only for single-byte characters.

Table 19: 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- and four-byte UTF-8	As separate words.

For more information on character-attribute tables, see Character attribute tables on page 43. For more information on modifying word-break tables, see Creating and modifying word-break tables on page 54. For more information on word-delimiter attributes, see Understanding word-delimiter attributes on page 54.

Guidelines for using multi-byte characters

When you use multi-byte characters in OpenEdge 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 enough space in the user interface for the IME.
- When using multi-byte characters and using a ABL 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 ABL element's reference entry in *ABL Reference*.

  To have the compiler warn you of ABL elements in the source code that should specify the unit of measure but appear not to, start OpenEdge with the Check Double-byte Enabled (`-checkdbe`) startup parameter. For more information on `-checkdbe`, see *Startup Command and Parameter Reference*.
- When displaying or printing characters, do not assume 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 OpenEdge, 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 Understanding Code Pages on page 27 and Using Multi-byte Code Pages on page 105. 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.

For details, see the following topics:

- Unicode overview
- Why use Unicode
- Using Unicode with OpenEdge products
- Using Unicode with OpenEdge databases
- Using Unicode with OpenEdge applications
- Guidelines for using Unicode
- Unicode support for supplementary characters
- ABL constructs that support Unicode filenames

Unicode overview

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, and so on—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, `http://www.unicode.org`. OpenEdge supports Unicode's UTF-8 encoding. In addition, all varieties of UTF-16 and UTF-32 are supported for input and output and for `LONGCHAR`s and CLOBs.

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.

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 frequently 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 multi-lingual data to a database and then display the data on client monitors and on printed reports.

The advantages of Unicode

Unicode has many advantages, as listed in the following table.

Table 20: Unicode's advantages

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.

Advantage	Explanation
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	OpenEdge clients that use incompatible code pages can easily read and write a single UTF-8 database.OpenEdge 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 multi-lingual.They can be deployed worldwide.
Interoperability	Active-X and Java clients are Unicode based.They can communicate with UTF-8 databases and AppServers.
Web compatibility	Unicode is becoming the universal code page of the Web. Current Web standards require Unicode and rely on it.
Multi-lingual applications	Applications using Unicode can support multiple languages in: • Data • User interface • Reports

Using Unicode with OpenEdge products

Some OpenEdge products can run directly in Unicode (that is, when `-cpinternal` is set to UTF-8). Other OpenEdge products cannot, but can run in a code page that OpenEdge can convert to and from Unicode.

The following products can run in Unicode:

- ActiveX client
- AppServer
- Progress Developer Studio for OpenEdge
- Batch client (GUI and character)
- Database server
- ABL compiler and r-code
- ABL GUI client (in interactive and batch modes)
- GUI Procedure Editor
- Java client
- SQL

Using Unicode with OpenEdge databases

Using UTF-8 with OpenEdge databases involves several techniques, described in the following sections:

- Converting an OpenEdge database to UTF-8 using the PROUTIL CONVCHAR utility on page 130
- Converting an OpenEdge database to UTF-8 using dump and load on page 131
- Compiling, storing, and applying the UTF-8 word-break rules to a database on page 132

Converting an OpenEdge database to UTF-8 using the PROUTIL CONVCHAR utility

This section describes the steps to convert an OpenEdge database to UTF-8 using the `PROUTIL CONVCHAR` utility.

To convert an OpenEdge 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:

```
proutil database-name -C convchar convert utf-8
```

2. Load the collation data definition (`.df`) file using the syntax for your specific operating system:

Windows syntax	`%DLC%\prolang\utf\filename.df`
UNIX syntax	**`$DLC`**`/prolang/utf/filename.df`

   For the UTF-8 BASIC collation, use the `_tran.df` collation data definition file. For an International Components for Unicode (ICU) collation, use one of the collation data definition files prefixed with "ICU" or "ICU_48" (such as `ICU-cs.df` used for Czech databases). For more information about character processing, see the README file in the directory OpenEdge/prolang.

3. Compile, store, and apply the UTF-8 word-break rules to your database. For complete instructions, see Compiling, storing, and applying the UTF-8 word-break rules to a database on page 132.
4. Rebuild the indexes using the following syntax:

```
proutil database-name -C idxbuild
```

Converting an OpenEdge database to UTF-8 using dump and load

This section describes the steps to convert an OpenEdge database to UTF-8 using dump and load.

To convert an OpenEdge database to UTF-8 using the dump and load utilities:

Caution: Before beginning, back up your database.

Do not use binary dump and load.

---

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 specific operating system:

Windows syntax	`prodb` *`database-name`* `%DLC%\prolang\utf\empty.db`
UNIX syntax	`prodb` *`database-name`* `$DLC/prolang/utf/empty.db`

Note: OpenEdge loads the UTF-8 BASIC collation (the `_tran.df` collation data definition file) in the empty UTF-8 database automatically and by default.

---

3. If you want to use an International Components for Unicode (ICU) collation, load the collation data definition (`.df`) file using the syntax for your specific operating system:

Windows syntax	`%DLC%\prolang\utf\`*`filename`*`.df`
UNIX syntax	`$DLC/prolang/utf/`*`filename`*`.df`

ICU collation data definition files are prefixed with "ICU" or "ICU_48" (such as `ICU-cs.df` used for Czech databases). For more information about character processing, see the README.txt file in the directory OpenEdge/prolang.

4. Compile, store, and apply the UTF-8 word-break rules to the database. For complete instructions, see Compiling, storing, and applying the UTF-8 word-break rules to a database on page 132.
5. 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.

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
- A QBW syntax error saying 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:

1. Compile a new version of the word-break table for UTF-8 using the syntax for your specific operating system, where *number* indicates an INTEGER between 1 and 255:

Windows syntax	`proutil -C wbreak-compiler %DLC%\prolang\convmap\utf8-bas.wbt` *number*
UNIX syntax	`proutil -C wbreak-compiler $DLC/prolang/convmap/utf8-bas.wbt` *number*

   This produces a new word-break table, proword.*number*.

2. Store the new word-break table using one of the following options:
   - 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:

```
proutil database-name -C word-rules number
```

Using Unicode with OpenEdge applications

You can create and run a character or graphical client application that uses Unicode. Since the OpenEdge database server and the OpenEdge graphical client support Unicode's UTF-8 encoding, but the OpenEdge interactive character client does not, OpenEdge applies a different set of rules for using Unicode with the batch character client and the graphical clients.

Rules for using Unicode with the OpenEdge character client

The following rules apply when using Unicode with the OpenEdge batch character client:

- An interactive character client must start up in a code page **other than** UTF-8.
- You must ensure a character client accesses only records in a compatible code page.
- You must follow guidelines for multi-byte programming, such as distinguishing characters, bytes, and columns.

Rules for using Unicode with the OpenEdge graphical client

You can use Unicode throughout your graphical application. When `-cpinternal` is set to UTF-8, the development and runtime environments for the ABL graphical client is fully Unicode-enabled.

Note: For the GUI client (prowin32.exe), text widgets such as editors and fill-ins are Unicode-enabled. If you enter characters into these widgets that are not supported by the font in use or are not available in the -cpinternal codepage, the widget value might contain a question mark (?).

The following rules apply when using Unicode with the OpenEdge graphical client:

- A graphical client may start up in the UTF-8 code page. If not, you must ensure the graphical client accesses only records in a compatible code page.
- You must follow guidelines for multi-byte programming, such as distinguishing characters, bytes, and columns.
- Specify which Unicode-enabled editor you want to use by setting the `UseSourceEditor` option in `progress.ini`. Set this option to NO to use RichEdit (which is fully Unicode-enabled) or YES to use the Source Editor with advanced editing features (the default). This setting applies to the Procedure Editor and the AppBuilder Section Editor; it does not apply to the ABL Editor widget.
- Use a Unicode font to display and print Unicode data. Specify a Unicode font by setting font options (such as `DefaultFont` and `PrinterFont`) in `progress.ini`.

Note: On Windows, you might also need to specify Unicode fonts on the Appearance tab in the Display Properties dialog box (accessed through the Windows Control Panel).

Unicode application example

Creating and running an application that uses Unicode is not difficult. The following is an example of creating and running an application consisting of a database server, a graphical client, and a UTF-8 database.

To create the application:

1. Convert the database to Unicode using one of the techniques in Using Unicode with OpenEdge databases on page 130.
2. Design queries that access only records that use the client's code page.

   One way to do this is for tables to have a field indicating 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 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 like the following:

```
FOR EACH Customer NO-LOCK WHERE SESSION:CPINTERNAL <> "UTF-8" OR
  db-language = user-language:
  DISPLAY Customer.Name Customer.Address Customer.City Customer.Country
    Customer.Comments.
END.
```

3. Start an OpenEdge 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.

Guidelines for using Unicode

When you use Unicode in OpenEdge applications, the following restrictions, cautions, and suggestions apply:

- With the OpenEdge UTF-8 BASIC collation, composed and decomposed characters are treated as different characters. With the International Components for Unicode (ICU) collations, composed and decomposed characters are treated as the same character for comparisons and indexes.
- The OpenEdge UTF-8 BASIC collation provides for sorting Unicode data in binary order. Alternatively, the ICU collations provide for sorting Unicode data based on the language-specific requirements for a locale.

Note: You can specify an OpenEdge collation or an ICU collation for sorting data using either the Collation Table (-cpcoll) startup parameter, or the COLLATE option on the FOR statement, the OPEN QUERY statement, and the PRESELECT phrase. For more information on the -cpcoll startup parameter, see *Startup Command and Parameter Reference*. For more information on the ABL elements, see *ABL Reference*.

For information about using ICU collations as database collations, see Using Databases on page 85.

- Before sorting Unicode data with the UTF-8 BASIC collation, normalize the data using the ABL NORMALIZE function. Normalizing the data converts the data into a standardized form that allows for more accurate and consistent sorting and indexing. This is important when working with characters or sequences of characters that have multiple representations (for example, base characters and combining characters) because it ensures that equivalent strings have a unique binary representation. For more information on the ABL NORMALIZE function, see *ABL Reference*.

Note: When sorting Unicode data with an ICU collation, you do not need to normalize the data.

- When UTF-8 data contains decomposed characters, you cannot convert it to a single-byte code page. You must first compose the data using the ABL `NORMALIZE` function. When you convert data from a single-byte code page to Unicode, the result is always composed data.
- OpenEdge 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 Understanding Code Pages on page 27 and Understanding Character Processing Tables on page 41.
- 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.

Unicode support for supplementary characters

OpenEdge supports *Unicode supplementary characters*. These are Unicode characters whose codepoints are in the supplementary planes 1-16; that is, codepoints from U+10000 to U+10FFFF. In UTF-8 encoding, these are 4-byte UTF-8 values, with lead bytes ranging from 0xF0 to 0xF4.

OpenEdge supports the UTF-16 and UTF-32 transformation formats as OpenEdge code pages that can be used for conversions. For example, in the following `ASC` and `CHR` functions:

```
ASC( '~U0254AE', "utf-32", "utf-8" )
CHR( 152750, "utf-8", "utf-32")
```

Note the decimal value of 0x254AE is 152750.

OpenEdge also supports UTF-16 for conversions. It converts supplementary characters, and conversions within `ASC` and `CHR` properly handle 2-byte values as unsigned shorts.

Using UTF-16 in the ASC and CHR functions

To use UTF-16 in the `ASC` function, use the following syntax:

Syntax

```
ASC( ch, "UTF-16" [, source-cp ] )
```

ch

One character, like `'A'`, `"A"`, or `'~U034e'`.

source-cp

The name of the source code page.

This returns one of the following values, which are platform independent:

- An integer less than 65536, representing the Unicode scalar value in plane 0
- A long integer composed of the high surrogate in the high-order 16 bits of the int and the low surrogate in the low-order 16 bits of the int

To use UTF-16 in the CHR function, use the following syntax:

Syntax

```
CHR( n, target-cp, "UTF-16" )
```

n

The numeric value of the character.

target-cp

The name of the target code page.

This command interprets its input as one of the following platform-independent values:

- The Unicode scalar value for plane 0
- An integer composed of a high surrogate in the high-order 16 bits and the low-surrogate in the low order 16 bits

When using UTF-16 in CODEPAGE-CONVERT, the behavior of character strings as UTF-16 are sensitive to byte order and the presence of null bytes. ABL can handle UTF-16 strings as RAW or MEMPTR data, and the use of PUT-UNSIGNED-SHORT and GET-UNSIGNED-SHORT solves any byte order machine dependencies.

Behavior of ASC and CHR functions

In the following example, assume -cpinternal is UTF-8 and the GUI code page is 1252:

```
ASC( '~U0254AE', "utf-32", "utf-8" )
```

This displays -191,978,590.

CHR accepts DECIMAL or INTEGER input values, so supplementary characters can be entered as positive values greater than the maximum integer ABL allows. The following examples display the multi-byte character consisting of the bytes 0xF4, 0x8E, 0xA3, and 0xA2:

```
DISPLAY CHR( 4102988706)
```

Or

```
CHR( -191978590 )
```

New and modified keywords

This section describes inputting Unicode codepoints in ABL code, in support of supplementary characters.

To input Unicode scalar codepoints in plane 0 (U+0000 to U+FFFF), use this syntax:

Syntax

```
~uXXXX in
```

XXXX

A 4-digit, case-insensitive hex digit.

To input Unicode scalar codepoints in planes 0 - 16 (U+0000 to U+10FFFF), use this syntax:

Syntax

```
~UXXXXXX
```

XXXXXX

A 6-digit, case-insensitive hex digit.

When the ABL code is parsed, this character value is converted from Unicode to `-cpinternal`. If the character is not a valid character in `-cpinternal`, the entire escaped string is passed through. For example, if `-cpinternal` is 1252, ~u4E00 is passed to ABL as is.

Limitations

Supplementary characters have the following restrictions:

- **Width** — Assume a column width of 2 for each supplementary character. This works for the HKSCS code page, but not for other supplementary characters
- **Word breaking** — Follow the heuristic rules already in place for UTF-8. Supplementary characters are considered as separate words, like 3-byte UTF-8 characters
- **Case rules** — No case transformations are applied to supplementary characters. This works for HKSCS, but not for other supplementary characters (in Unicode 3.1, Deseret characters are the only supplementary characters with case mappings)

ABL constructs that support Unicode filenames

A subset of ABL constructs support the use of the full Unicode character set in filenames, including both ASCII and non-ASCII characters.

The following table lists those constructs and the associated elements that can contain any Unicode character. Windows natively stores filenames as Unicode (UTF-16), and for these constructs, the AVM converts filenames between Unicode and the code page set in the startup parameter `-cpinternal`. Most UNIX/Linux implementations support UTF-8 Unicode filenames. In both Windows and UNIX/Linux systems, the parameter `-cpinternal` should be set to UTF-8 to ensure that the AVM will be able to properly access files with Unicode names. For more information on `-cpinternal`, see *Startup Command and Parameter Reference*.

Table 21: ABL constructs that support Unicode filenames

ABL construct	Elements that can accept Unicode values
`ADD-SCHEMA-LOCATION`	*location*
`CONTEXT-HELP-FILE`	This attribute can hold pathnames that contain Unicode characters.
`COPY-LOB`	`FROM` *source-filename* `TO` *target-filename*
`DEFINE IMAGE`	`FILE` *name*
`DOS`	*command-token* | `VALUE (` *expression* `)`
`FILE-NAME`	When used with the `FILE-INFO` handle, this attribute can hold filenames that contain Unicode characters.
`FILE-FULL-PATHNAME`	This attribute can hold pathnames that contain Unicode characters.
`FILE-PATHNAME`	This attribute can hold pathnames that contain Unicode characters.
Framephrase, `CONTEXT-HELP-FILE`	*help-file-name*
`GET-DROPPED-FILE`	This method can return pathnames that contain Unicode characters.
Image phrase, `FILE`	*name*
`INPUT FROM`	*opsys-file* | `VALUE (` *expression* `)` `OS-DIR (`*directory*`)`
`INPUT THROUGH`	*program-name* `VALUE (` *expression* `)`
`LOAD` (statement)	`DIR` *directory*
`LOAD` (handle method)	*file*
`LoadControls`	*control-filename*
`LOAD-ICON`	*icon-filename*
`LOAD-IMAGE`	*filename*
`LOAD-MOUSE-POINTER`	*pointer-name*

ABL construct	Elements that can accept Unicode values
LOAD-PICTURE	*image*
LOAD-SMALL-ICON	*smallicon-filename*
OS-APPEND	source-filename | VALUE (expression) target-filename | VALUE (expression)
OS-COMMAND	command-token | VALUE (expression)
OS-COPY	source-filename | VALUE (expression) target-filename | VALUE (expression)
OS-COMMAND	command-token | VALUE (expression)
OS-CREATE-DIR	dirname | VALUE (expression)
OS-DELETE	filename | VALUE (expression)
OS-RENAME	source-filename | VALUE (expression) target-filename | VALUE (expression)
OUTPUT THROUGH	program-name | VALUE (expression)
OUTPUT TO	opsys-file | VALUE (expression)
PATHNAME	This attribute can hold pathnames that contain Unicode characters.
PRINTER-NAME	This attribute can hold printer names that contain Unicode characters.
PROCEDURE	EXTERNAL " *dllname*"
READ-FILE	*filename*
READ-XML	*file schema-location*
READ-XMLSCEHMA	*file*
SAVE	*file*
SAVE-FILE	*filename*
SCHEMA-LOCATION	This attribute can hold values that contain Unicode characters.

ABL construct	Elements that can accept Unicode values
SCHEMA-PATH	This attribute can hold pathnames that contain Unicode characters.
SEARCH	*opsys-file*
SET-INPUT-SOURCE	*file*
SET-OUTPUT-DESTINATION	*file*
SYSTEM-DIALOG-GET-DIR	*character-field* INITIAL-DIR *directory-string*
SYSTEM-DIALOG-GET-FILE	*character-field* INITIAL-DIR *directory-string*
SYSTEM-HELP	*file-string*
WRITE-XML	*file schema-location*
WRITE-XMLSCHEMA	*file*
XML-SCHEMA-PATH	This attribute can hold pathnames that contain Unicode characters.

The ABL constructs listed in the following table do not support the full Unicode character set. Some constructs have strict limits, as noted in the table, but the rest are limited according to the operating system. In Windows, they are restricted to characters from the code page set in the Language for non-Unicode Programs (which is found under Regional Settings in the Control Panel for Windows 7). A total of 256 characters can be used in Windows filenames, including non-ASCII characters, but the code page set in `-cpinternal` must match the code page selected in the Language for non-Unicode Programs. If your filenames contain Unicode characters outside of that code page, the AVM will not be able to access the file. In UNIX/Linux, the set of characters should be restricted to those available in the code page set in `-cpinternal`.

For both Windows and UNIX/Linux, `-cpinternal` must always be the same if a filename used with one of the constructs in the following table contains characters outside of ASCII. Restricting all filenames used with these constructs to ASCII characters only (unless a tighter restriction is noted) is the recommended practice.

Table 22: ABL constructs that only support restricted character sets

{ } (include)	CALL-NAME
CLASS[2]	COMPILE
CONNECT[3]	CREATE DATABASE[3]
INTERFACE[2]	LIBRARY[2]
LOG-MANAGER:LOGFILE-NAME	NEW[2]
PROMSGS	PROPATH

[2] Restricted to alphanumeric characters plus the following: # $ % & _

[3] Restricted to alphanumeric characters only. Diacritical marks and the following characters are not permitted: \ " ‘ * ; | ? [ ] ( ) ! { } < > @ + = : ~

RCODE-INFO[4]	RUN
SAVE-CACHE	SESSION:BASE-ADE
USING[2]	

[4] Note that FILE-NAME, when used with the FILE-INFO handle, does support Unicode characters, but when FILE-NAME is used with RCODE-INFO, it does not.

10

Deployment and Configuration

OpenEdge 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, and others you modify by using Progress Explorer. Also, many OpenEdge products provide parameters that let you localize the output.

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 Understanding Character Processing Tables on page 41 and Understanding Code Pages on page 27.

For details, see the following topics:

- The progress.ini file and the Windows registry
- The PROTERMCAP file
- The PROMSGS file
- Parameter files
- The conmgr.properties file
- Report Builder parameters
- Additional guidelines

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.

OpenEdge 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 OpenEdge 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 OpenEdge `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 *Manage ABL Applications* and *Configure OpenEdge*.

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]` and the `[fonts]` sections.

[Startup]

Some `progress.ini` file settings you can localize appear in the `[Startup]` section. The following figure shows the `[Startup]` section of the `progress.ini` file that OpenEdge provides in the `OpenEdge/bin` directory.

[Startup] section of the progress.ini file

```
[Startup]
V6Display=no
;ImmediateDisplay=yes
;MultitaskingInterval=100
DefaultFont=MS Sans Serif, size=8
DefaultFixedFont=Courier New, size=8
```

The following table describes the `[Startup]`section settings you might localize.

Table 23: 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 ABL `OUTPUT TO` statement's `PRINTER` option.

Setting	Description
PROMSGS	The name of the promsgs file the application uses. For example, to use the Swedish PROMSGS file, set PROMSGS to OpenEdge\prolang\swe\promsgs.swe
PROSTARTUP	The name and path of the startup parameter (.pf) file OpenEdge expects.

[fonts]

Other progress.ini file settings you might localize appear in the [fonts] section. The following figure shows the [fonts] section of progress.ini that OpenEdge provides in the OpenEdge/bin directory.

[fonts] section of the progress.ini file

```
[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
```

Progress Software Corporation ships progress.ini with native font names as the default. Users who download support for other languages might need to replace the native font names with English font names. For example, the following lines in the [fonts] section of progress.ini allows the user to view Chinese on systems running the English version of Microsoft Windows:

```
[fonts]
DefaultFont=MS Song, size=9 Script=gb2312, fixed
...
font6=MS Song, size=9 Script=gb2312
...
```

Note the font name is in Latin characters.

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. 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.

The PROTERMCAP file

The PROTERMCAP file, used by UNIX character-interface applications only, tells OpenEdge how to interact with a particular terminal, console, or terminal emulator. For more information on the PROTERMCAP file, see *Manage ABL Applications*.

The PROMSGS file

The PROMSGS file contains OpenEdge 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 OpenEdge/prolang/hun directory, and the Japanese PROMSGS file, promsgs.jpn, resides in the OpenEdge/prolang/jpn directory.

OpenEdge 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 OpenEdge 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 Using SQL on page 93

Parameter files

OpenEdge 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. OpenEdge itself supplies such a set of parameter files. These reside in the `OpenEdge/prolang` directory in subdirectories for each language. For example, the Russian parameter file, `russian.pf`, resides in the `OpenEdge/prolang/rus` directory. Here is the Russian parameter file:

```
-d dmy
-lng "Russian"
-cpcase Basic
-cpcoll Russian

# -cpinternal - 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 - 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, OpenEdge ignores lines that start with the pound sign (#).

The startup parameters you might localize appear in the following table.

Table 24: Startup parameters that can be localized

Startup parameter		Description
`-cpcase`	Case table	The name of the case table used by the OpenEdge database server and the OpenEdge client
`-cpcoll`	Collation table	The name of a collation table within the `convmap.cp` file, or the name of a collation in the International Components for Unicode (ICU) library, used by the OpenEdge database server and the OpenEdge client
`-cpinternal`	Internal code page	The name of the code page used internally by the OpenEdge database server and the OpenEdge client
`-cpstream`	Stream code page	The name of code page OpenEdge uses for stream I/O
`-d`	Date format	The format in which OpenEdge displays dates in the application
`-lng`	Language	The value that the `CURRENT-LANGUAGE` function returns.
`-numdec`	Fractional separator	The character that separates the integer portion and the fractional portion of a decimal number
`-numsep`	Thousands separator	The character that separates each group of three digits in the integer portion of a number

You can also use parameter files with OpenEdge utilities, such as `PROSHUT` and `PROUTIL`. For more information on these utilities, see *Manage the OpenEdge Database*. For more information on startup parameters, see *Startup Command and Parameter Reference*.

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 the following table.

Table 25: 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 OpenEdge database server and the OpenEdge client	BASIC
collationtable	-cpcoll	The name of a collation table within the convmap.cp file, or the name of a collation in the International Components for Unicode (ICU) library, used by the OpenEdge database server and the OpenEdge client	BASIC
conversionmap	-convmap	The name and path of the compiled CONVMAP file used by the OpenEdge database server and the OpenEdge client	*OpenEdge*/convmap.cp
internalcodepage	-cpinternal	The name of the code page used internally by the OpenEdge database server	ISO8859-1
logcharacterset	-cplog	The name of the code page used by the OpenEdge database's log file	ISO8859-1

Note: Do not edit the conmgr.properties file directly. Instead, use Progress Explorer. For more information on Progress Explorer, see its online help.

For more information on the conmgr.properties file, see *Manage the OpenEdge Database*.

Report Builder parameters

The OpenEdge Report Builder provides parameters that let you localize its output. The following table describes these parameters.

Table 26: 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 and the integer portion of a number

Note: OpenEdge includes only the run-time component of Report Builder.

Additional guidelines

When deploying and configuring an application across multiple locales, the guidelines in this section might save you time and trouble.

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 OpenEdge, enter the following command at the command prompt:

```
stty -istrip
```

Specifying -cpinternal and -cpstream with database utilities

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

A

OpenEdge Resources

This appendix describes the resources OpenEdge provides for internationalization and localization.

For details, see the following topics:

- Files in the OpenEdge/prolang directory
- Startup parameters and settings
- ABL
- OpenEdge SQL
- Utilities
- Determining the code page

Files in the OpenEdge/prolang directory

The `OpenEdge/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, `OpenEdge/prolang/rus` contains files for Russian and `OpenEdge/prolang/tai` contains files for Thai. For more information on the subdirectories and their contents, see the `OpenEdge/prolang/readme` file.

The following table describes the files in the `OpenEdge/prolang` directory. In the table, *`lng`* represents a three-letter abbreviation for a particular locale.

Table 27: Files in OpenEdge/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 because some locales that share a language use different code-pages, collation orders, or date formats.
lng/promsgs.lng	Contains OpenEdge error messages for a particular locale. Set the PROMSGS environment variable to the appropriate file. For example, PROMSGS=%DLC%\prolang\sch\promsgs.schcauses an application to display OpenEdge error messages in Simplified Chinese.

Startup parameters and settings

The following table lists startup parameters useful when deploying applications across multiple locales. For more information on these startup parameters see *Startup Command and Parameter Reference*.

Table 28: Startup parameters

Syntax	Name	Description
-checkdbe	Check double-byte enabled	Causes OpenEdge to report a compile-time error whenever it finds a LENGTH function, SUBSTRING function, or OVERLAY statement that does not use the CHARACTER, COLUMN, FIXED or RAW option.Use -checkdbe when you modify an OpenEdge application to support double-byte or Unicode characters.
-convmap	Conversion map	Specifies the conversion map file to use for code-page conversions, collation orders, and case conversions.The default conversion map file is OpenEdge/convmap.cp.
-cpcase	Case table	Specifies a case table in the convmap.cp file to use for case conversions.The default is BASIC. OpenEdge performs case conversions when you use the CAPS and LC functions, and when you use "!" in a character-field format string, which tells OpenEdge to convert all characters in the string to uppercase during input.
-cpcoll	Collation table	Specifies a collation table in the convmap.cp file, or a collation in the International Components for Unicode (ICU) library, to use for sorting.The default is BASIC.
-cpinternal	Internal code page	Specifies the code page OpenEdge uses in memory and for graphical clients.The default is the ISO8859-1 code page.
-cplog	Log file code page	Specifies the code page OpenEdge uses when it writes to a log file.The default is the code page specified by -cpstream.
-cpprint	Print code page	Specifies the code page OpenEdge uses when it prints to a printer.The default is the code page specified by -cpstream.
-cprcodein	R-code in code page	Specifies the code page OpenEdge uses when it reads r-code text segments.The default is the code page described in the r-code.
-cprcodeout	R-code out code page	Specifies the code page OpenEdge uses when it writes r-code text segments.The default is the code page specified by -cpinternal.
-cpstream	Stream code page	Specifies the code page for stream I/O.The default code page is IBM850.
-cpterm	Terminal code page	Specifies the code page for output to a character terminal.The default is the code page specified by -cpstream.
-d	Date format	Specifies the date format; that is, the order in which the day, month, and year appear. Specify the format as a three-character string, comprising the letters *d* (day), *m* (month), and *y* (year), in the preferred

Syntax	Name	Description
		order.The default is "*mdy*".The -d startup parameter corresponds to the DATE-FORMAT attribute of the SESSION system handle.
-E	European numeric format	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.
-isnoconv	Initial value segment no convert	Disables a code-page conversion that was not provided in earlier versions of OpenEdge.
-lng	Language	Specifies the initial value for the CURRENT-LANGUAGE function, which determines the r-code segment from which OpenEdge reads character-string constants.Specify the language as a quoted character string.
-numdec	Fractional separator	Specifies the character that separates the integer portion and the fractional portion of a decimal number.
-numsep	Thousands separator	Specifies the character that separates each group of three digits in a number.
-yr4def	Four-digit year default	Outputs a four-digit year from EXPORT MESSAGE and PUT UNFORMATTED statements that might use a two-digit-year.
-yy	Century year offset	Specifies the starting year for the OpenEdge two-digit year range of 100 years.The -yy startup parameter corresponds to the YEAR-OFFSET attribute of the SESSION system handle.

The following table lists the OpenEdge Explorer or OpenEdge Management properties you can localize. For each property, the table gives the corresponding startup parameter.

Table 29: OpenEdge Explorer and OpenEdge Management properties that can be localized

OpenEdge Explorer setting	OpenEdge 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
Conversion map	`OpenEdge/convmap.cp`	`-convmap`	`OpenEdge/convmap.cp`

ABL

The following table describes the ABL elements particularly useful for internationalizing and localizing applications. For more information, see *ABL Reference*.

Table 30: ABL elements that support internationalization and localization

ABL element	Description
`ASC` function	Returns the value that a character has in the code page OpenEdge uses (default or the code page that `-cpinternal` specified). You can also specify a target and source code page, so `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, OpenEdge uses a case table. Any double-byte characters in a double-byte code page are not changed.

ABL element	Description
CHR function	Returns the character that an integer represents in the code page OpenEdge uses (default or the code page that -cpinternal specified). You can also specify a target and source code page, so that CHR returns the value in the source code page and returns the character that the value represents in the target code page. **Note:** Do not use CHR 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 on the FOR statement, OPEN QUERY statement, and PRESELECT phrase	Generates the collation value of a string after applying a particular strength, and optionally, a particular collation.
COMPARE function	Compares two strings after applying a particular strength, and optionally, a particular collation.
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 OpenEdge 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 OpenEdge session uses.
GET-COLLATIONS function	Returns the list of collation tables in the convmap.cp file that the current OpenEdge session uses.
LANGUAGES option of the COMPILE statement	Specifies which languages to generate text segments in r-code. At compilation, OpenEdge reads translated strings from the translation databases that you provide. The r-code has a text segment for each language.
LC function	Changes any uppercase letters in a character string to lowercase. OpenEdge 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.

ABL element	Description
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.
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 OpenEdge session uses.
SET-NUMERIC-FORMAT method of the SESSION system handle	Sets the NUMERIC-SEPARATOR and NUMERIC-DECIMAL attributes to arbitrary values simultaneously.

The following table describes the ABL elements that support multi-byte characters.

Table 31: ABL elements that support multi-byte characters

ABL element	Description/Multi-byte support
APPLY statement	Performs the function of a specified key. The expression you APPLY can contain multi-byte characters.
ASC function	You can specify source and target code pages. If you do, ASC converts the value from source to target before returning it to you. Code pages can be multi-byte. For a double-byte character, ASC returns a value between 256 and 65,535. For a UTF-8 character up to 3 bytes, ASC returns a value between 65,536 and 15,712,191. For a Unicode supplementary character (4 bytes), ASC returns a negative integer between -191,905,857 and -258,965,376.For an invalid character, ASC returns -1.
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 a double-byte code page are not changed. Any UTF-8 character up to 3 bytes is subject to upper easing.

ABL element	Description/Multi-byte support
CHR function	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. For values between 256 and 44,103,061,439, inclusive, CHR returns a character, if the value represents a valid combination of lead and trail bytes. For values not representing valid combinations of lead and trail bytes, CHR returns a null string. For double-byte characters, you can run CHR on one byte at a time.For Unicode supplementary characters, CHR can accept as input: • The large decimal values obtained from the 4-byte UTF-8 characters ranging from 0xF0809090 to 0xF48FBFBF • The negative integer values returned by ASC for Unicode supplementary characters • Hex values (such as 0xF0809090)
CODEPAGE-CONVERT function	Converts a string value from one code page into another.
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 ENCODE function reference entry in *ABL 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.
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.
FORMAT phrase	Specifies the display format of, among other things, character strings. For a field containing multi-byte characters, you can specify the width in columns.
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.

ABL element	Description/Multi-byte support
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. You can set the unit of measure to characters, bytes, or columns.
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.
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. You can set the unit of measure to characters, bytes, or columns.
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 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. (A key sequence is the set of keystrokes necessary to send one character or function key to the application.)
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 source value of any data type to a character value. The source value can contain multi-byte characters.
SUBSTITUTE function	Returns a character string that is made up of a base string plus substitute arguments in the string. The base string and arguments can contain multi-byte characters.
SUBSTRING function	Extracts a position of a character string from a field or variable. You can set the unit of measure to characters, bytes, or columns.

ABL element	Description/Multi-byte support
`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. You can set the unit of measure to characters, bytes, or columns.
`TRIM` function	Removes leading and trailing single-byte spaces in a character string. The string and trimmed characters can contain multi-byte characters.

OpenEdge SQL

This section describes the resources OpenEdge SQL provides for internationalization and localization.

SQL elements that support internationalization and localization

The following table describes the OpenEdge SQL elements that support internationalization and localization.

Table 32: OpenEdge SQL language elements that support internationalization and localization

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.
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* can 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.

Syntax	Description	Comment
`'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 can 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.

`{ 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 only in the English language.

`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.

`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. Otherwise, `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.	

Syntax	Description	Comment
		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. Otherwise, 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.
CONVERT ('datatype [(length)] ', expression)	The ABL 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, OpenEdge uses a case table in the convmap.cp file. The default case table is BASIC.
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.

Syntax	Description	Comment
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 can contain multi-byte characters. The conversion to lowercase conversion depends on the case table in the convmap file. The default case table is BASIC.
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* can 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* can 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.

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.
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 sort weights in the collation table in the database.

Syntax	Description	Comment
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.
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 and 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 and depend on sort weights in the collation table in the database.
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.

Format specifiers allowed with the TO_CHAR and TO_DATE functions

The following table describes the format specifiers you can use with the SQL `TO_CHAR()` and `TO_DATE()` functions.

Table 33: Format specifiers used with TO_CHAR() and TO_DATE()

Format specifier	Description	Comment
`$`	A value with a leading dollar sign. For example: `$999`.	-
`,`	A value with a comma in a specified position, but not a thousands separator. For example: `1.000,00`.	-
`.`	A value with a decimal point in a specified position. For example: `99.99`.	-
`0`	A value displaying a leading or trailing zero. For example, `0999` or `9990`.	-
`9`	A set number of significant digits to be displayed. Display leading space if positive, leading minus if negative. Leading zeros are blank, except for a zero value returning a zero for the integer part of the number. For example: `9999`.	-
`A.M.`	Meridian indicator in the native language—with periods.	English only
`AM`	Meridian indicator in the native language—without 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, and so on.
`D`	The day of the week as a one-digit number between 1 and 7.	-
`D`	A value with an `NLS_NUMERIC_CHARACTER` in the specified position. The default `D` character is a period (`.`), for example: `99D9`.	-
`day`	The day of the week in the native language—first letter only in uppercase.	English only
`Day`	The day of the week in the native language—first letter only in uppercase.	English only
`DAY`	The day of the week in the native language—entire name in uppercase.	English only
`DD`	The day of the month as a two-digit number between 01 and 31.	-

Format specifier	Description	Comment
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
G	A value with an NLS_NUMERIC_CHARACER in the specified position. The default G character is a comma (,), for example: 9G999.	-
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.	-
L	A value with the local currency symbol NLS_CURRENCY in the specified position. The default L character is a dollar sign ($). For example: L999.	-
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
MONTH	The first nine characters of the name of the month in the native language, right-padded with blanks.	English only
P.M.	Meridian indicator in the native language—with periods.	English only
PM	Meridian indicator in the native language—without periods.	English only
Q	The quarter of the year as a single digit between 1 and 4.	-

Format specifier	Description	Comment
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, and so on.
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.	-
Y,YYY	The year as a four-digit number—comma separates the first digit from the other digits.	-
YY	The year as a two-digit number.	-
YYY	The year as a three-digit number.	-
YYYY	The year as a four-digit number.	-

Utilities

The following table lists the utilities mentioned in this book.

Table 34: 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 *Manage the OpenEdge Database*. For the complete syntax of PROLIB, see *Manage ABL Applications*.

Determining the code page

When OpenEdge 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. The following table lists typical application components and for each, and describes how OpenEdge determines the code page and how the developer can determine the code page.

Table 35: Determining the code page

Application component	How OpenEdge determines the code page	How a developer can determine the code page
Client or server memory (applies to the following executables: _progres, _prosrv, _mproshut, _proutil)	OpenEdge 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	OpenEdge 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	OpenEdge 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.
PROMSGS files	OpenEdge 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...	OpenEdge 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	OpenEdge checks the value of the -cpterm parameter, and if no value is specified for this parameter, OpenEdge 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	OpenEdge 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.

Application component	How OpenEdge determines the code page	How a developer can determine the code page
R-code (`.r`) files	OpenEdge 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 dump (`.d`) and data definition (`.df`) files	OpenEdge 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.

B

Character Processing Table Formats

This appendix describes the format of the character processing tables.

For details, see the following topics:

- Character attribute table
- Case table
- Collation table
- Code-page conversion table
- Word-break table

Character attribute table

The following figure 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*/ 000 000 000 000 000 000 001 001 001 000 000 000 000 000 000 000
  /*192-207*/ 000 000 000 000 000 000 000 001 001 000 000 000 000 000 000 000
  /*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
```

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 OpenEdge 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 in the figure.

The `ENDTABLE` keyword signals the end of the character attribute table. The `ENDCODEPAGE` keyword signals the end of the table entry.

Case table

The following figure 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
```

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 in the figure.

The `ENDTABLE` keyword signals the end of the character attribute table. The `ENDCASE` keyword signals the end of the table entry.

Collation table

The following figure 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
```

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 OpenEdge 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 each. Be sure to format the rows of data exactly as shown in the figure.

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.

Code-page conversion table

The following figure 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
```

The CONVERT keyword signals the beginning of a code-page conversion table entry. The NOINVERSE option of CONVERT tells OpenEdge not to create the table for the inverse conversion. If NOINVERSE does not appear, OpenEdge 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 in the figure.

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, OpenEdge 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. The following example shows part of the ISO 8859-1 to IBM850 conversion table. To perform the conversion, OpenEdge 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
```

Convmap changes

The convmap files support supplementary characters in the DBCS-to-UTF-8 conversion (type 17). Any conversion using supplementary characters must use the Unicode compression option. Unicode compression is input as an increasing set of Unicode pairs of the form high value, low value, followed by 0, for a maximum of 5 ranges, with an additional 0 at the end.

The following excerpt from a conversion specification compresses the Unicode values from four ranges in plane 0 and one range in plane 8:

```
CONVERT
SOURCE-NAME "SUPPLEMENTARY-CP"
TARGET-NAME "UTF-8"
TYPE "17"
FIRST-CONSTANTS
# single, 1st start, 2nd start, gap
0xFFFF 0 0x580 0x4ABF
# ccount, umin, umax, index power, delta power, bad character
779 0x0 0x0FFD 6 6 0
# unicode compression
0x451 0xA7 0 0x266F 0x2000 0 0x33CD 0x3000 0 0xFA2D 0xF900 0 0x8FFE5 0x8FF00 0 0

# High range
0x00 0x00 0x81 0xFC
# low range
0x40 0x7E 0x80 0xFC 0xFF 0xFE
ENDCONSTANTS
```

Compression cannot occur over more than one plane. For example, this would not be supported:

```
0x451 0xA7 0 0x345EE 0x20000 0 0 0 0 0 0 0 0 0 0 0
```

The actual mappings appear in the -DATA section of the table, with the Unicode values expressed as the hex value of the UTF-32 codepoint, as shown:

```
FIRST-DATA
0x00D8      0x8FF98
0x00DF      0x8FF9F
0x8140      0x3000
0x8141      0x3001
0x8143      0x8FF0C
0x8144      0x8FF0E
ENDTABLE
```

Word-break table

The syntax for the word-break table:

Syntax

```
[ #define symbolic-namesymbol-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 OpenEdge lets you compile word-break tables that omit all items within the second pair of square brackets, Progress Software Corporation recommends 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, Progress Software Corporation recommends 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: OpenEdge allows a table type of 1 or 2. Although these are still supported, Progress Software Corporation 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.

The following table describes the word-delimiter attributes.

Table 36: Word delimiter attributes

Word delimiter attribute	Description	Default
LETTER	Always part of a word	Assigned to all characters 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 numerals 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; otherwise, 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.
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

_tran.df file
    example
    50–51

7-bit characters
    code page of *36*
8-bit characters
    code page of *36*

A

Accelerators
    with multi-byte labels *126*
ActiveX clients *129*
Addresses
    international formats *68*
    postal codes
        CHARACTER data type *68*
AppBuilder
    multi-byte support *107*
    multiple layouts *82*
Application Compiler
    multi-byte support *107*
APPLY statement
    multi-byte support *157*
AppServers
    multi-byte support *107*
ASC function
    multi-byte support *157*
ASCII characters
    code page of *36*
ASCII function *162*

B

Batch clients
    multi-byte support *108*
BEFORE_DIGIT
    word-delimiter attribute *183*
BEFORE_LET_DIG
    word-delimiter attribute *183*
BEFORE_LETTER
    word-delimiter attribute *183*
BEGINS operator
    multi-byte support *157*
BULKLOAD qualifier
    PROUTIL utility *171*

C

Calendars
    international conventions *67*
CAPS function
    multi-byte support *157*
Carons *87*
Case Code Page (-cpcase) startup parameter *71*, *153*
Case conversions *71*
CASE statement *62*
Case tables
    format *177*
CASE-INSENSITIVE-SORT collation tables *51*, *69*
CASE-SENSITIVE-SORT collation tables *51*, *69*
casetablename property
    conmgr.properties file *149*
Celsius (temperature scale) *66*
Century Year Offset (-yy) startup parameter *154*
CHAR function *162*
Character attribute tables
    format *176*
Character clients
    multi-byte support *108*
Character conversions
    Data Dictionary rules *173*
    OpenEdge executables *173*
Character processing
    byte level *62*
Character processing tables
    definition of *41*
Character sets
    definition of *23*, *30*
Character terminals
    code page of *37*
Character-string literal *162*
CHARSET attribute
    SESSION handle *172*
Check Double-byte Enabled (-checkdbe) startup parameter *153*
-checkdbe startup parameter *153*
chkdot.p utility *36*
CHR function
    multi-byte support *158*
    sorting *69*
Clients
    multi-byte support *108*
Code pages
    character terminals *172*
    client or server memory *172*
    data definition (.df) files *173*
    databases *172*

Code pages *(continued)*
definition of *28*
graphical devices *172*
procedure (.p) files *172*
PROMSGS file *172*
r-code (.r) files *173*
table dump (.d) files *173*
undefined, definition of *35*
Code points
definition of *28*
Code-page conversion
character conflicts
scanning for *89*
databases *90*
diagram *33*
Code-page conversion tables
format *179*
Code-page trailers *91*
CODEPAGE-COMPILER qualifier
PROUTIL utility *171*
CODEPAGE-CONVERT function *156*, *158*
COLLATE option *130*, *134*, *156*
Collation Code Page (-cpcoll) startup parameter *153*
COLLATION keyword
collation tables *51*
Collation tables
CASE-INSENSITIVE-SORT *51*
CASE-SENSITIVE-SORT *51*
_tran.df file *152*
dumping *50*
editing *51*
format *178*
loading *52*
modifying *50*
Collations
indexes *70*
international *69*
collationtable property
conmgr.properties file *149*
Colors
selecting *76*
COMPARE function *156*
COMPILE statement
LANGUAGES option *156*
Composed characters
definition of *130*, *134*
CONCAT function *163*
CONTAINS operator
WHERE option
record phrase *53*
CONV89 qualifier
PROUTIL utility *171*
CONVCHAR qualifier
PROUTIL utility *171*
Conversion Map (-convmap) startup parameter *153*
conversionmap property
conmgr.properties file *149*
Conversions
case *71*
CONVERT function *163*
-convmap startup parameter *93*, *153*
convmap.dat file
compiling *49*
modifying *49–50*
providing access to *50*
-cpcase startup parameter *71*, *93*, *148*, *153*
-cpcoll startup parameter *93*, *148*, *153*
-cpinternal startup parameter *92–93*, *153*
CPINTERNAL attribute
SESSION handle *172*
-cplog startup parameter *93*, *153*
-cpprint startup parameter *153*
-cprcodein startup parameter *153*
-cprcodeout startup parameter *153*
CPSTREAM attribute
SESSION handle *172*
-cpstream startup parameter *92–93*, *148*, *153*
CPTERM attribute
SESSION handle *172*
-cpterm startup parameter *153*
Currencies
data types *66*
international formats *76*
CURRENT-LANGUAGE function *62*, *156*
CURRENT-LANGUAGE statement *156*

D

-d startup parameter *67*, *153*
Data definition (.df) files *34*
Data Dictionary
multi-byte support *108*
Data types
INTEGER *67*
Database collation tables *69*
Database names
restrictions *85*
Database servers *93*
Databases
collation *70*
empty *86*, *152*
DataServers
multi-byte support *108*
Date data
storing *67*
DATE data type *67*
Date Format (-d) startup parameter *67*, *148*, *153*
DATE function *67*
Date-format literals *162*

Dates
handling international standards
sample procedure 67
international conventions 67
processing 67
DAY function 67
DBCODEPAGE function
4GL 37
DBCOLLATION function 156
DECIMAL data type
display format 65
Decimal point
numeric formats 65
Diereses 87
DIGIT
word-delimiter attribute 183
Display formats 112
Double-byte characters
collation order 124–126
definition of 106
sorting 124–125
Double-byte code pages
definition of 106
Double-byte enabled
LENGTH function 117

E

-E startup parameter 65, 154
ENCODE function
multi-byte support 158
ENTRY function
multi-byte support 158
ENTRY statement
multi-byte support 158
multi-byte support of 158
Euro symbol 87
European Numeric Format (-E) startup parameter 65, 154
Explorer
OpenEdge 93

F

Field labels
localized 77
FILL function
multi-byte support 158
Fonts
setting in progress.ini 145
FOR statement
COLLATE option 156
FORMAT option
displaying dates 67
FORMAT phrase
multi-byte support 158
Four-Digit Year Default startup parameter 154
Fractional separator
numeric formats 65
Fractional Separator (-numdec) startup parameter 148, 154

G

Gaiji characters 123
GET-CODEPAGES function 156
GET-COLLATIONS function 156
Graphical clients
multi-byte support 108
GREATEST function 163

H

HLC Applications
multi-byte support 108

I

ICU collations 46
IDXBUILD qualifier
PROUTIL utility 171
IDXFIX qualifier
PROUTIL utility 171
IGNORE
word-delimiter attribute 183
Images
cross-cultural 78–79
IME, See Input Method Editors
INCLUDE directives 42
INDEX function
multi-byte support 158
Indexes
rebuilding for collation changes 52
rebuilding for word-break changes 58
INI2REG utility 144
INITCAP function 163
Initial Value Segment No Convert (-isnoconv) startup parameter 154
Initialization (.ini) files 144
INPUT FROM statement
4GL 34
Input method editors
definition of 109
Input methods
definition of 109
INSERT function 163
Installation program
OpenEdge 144
INSTR function 164
INTEGER data type 67

Internal Code Page (-cpinternal) startup parameter *148*, *153*
internalcodepage property
  conmgr.properties file *149*
Internationalized applications
  definition of *22*
IS-LEAD-BYTE function
  multi-byte support *158*
-isnoconv startup parameter *154*

J

Java clients *129*

K

Keyboards
  character sets *74*
  mapping *64*

L

Language (-lng) startup parameter *148*, *154*
Language issues
  character sets *77*
  direction text is read *77*
  input method editors *77*
  translation *83*
  user interface text *77*
LANGUAGES option
  COMPILE statement *156*
LASTKEY function
  multi-byte support *158*
Layout, See Screen layout
LC function
  multi-byte support *159*
LCASE function *164*
Lead bytes
  definition of *106*
LEAST function *164*
LEFT function *164*
LEFT-TRIM function
  multi-byte support *159*
LENGTH function *117*, *159*, *164*
LETTER
  word-delimiter attribute *183*
Libraries
  Progress
    code page of *39*
Ligatures *87*
LIKE predicate *161*
-lng startup parameter *154*
Locale
  definition of *22*
Localization
  graphics *78*
  multiple layouts *82*
  user interface text *74*
Localized application
  definition of *22*
LOCATE function *164*
Log File Code Page (-cplog) startup parameter *153*
logcharacterset property
  conmgr.properties file *149*
LOOKUP function
  multi-byte support *159*
LPAD function *165*
LTRIM function *165*

M

MATCHES operator
  multi-byte support *159*
Metric system *66*
Monitors
  different models *74*
MONTH function *67*
Multi-byte characters
  definition of *106*
Multi-byte code pages
  definition of *106*
Multi-lingual data *128–129*
Multiple layouts *82*

N

N-byte characters
  definition of *106*
N-byte code pages
  definition of *106*
Negative numbers
  international notation *66*
Non-ASCII characters *85*
Normalization
  Unicode *134*
NUM-ENTRIES function
  multi-byte support *159*
Numeric formats
  internal processing *65*
  international standards *65*
NUMERIC-DECIMAL-POINT attribute
  SESSION handle *156*
NUMERIC-FORMAT attribute
  SESSION handle *65*, *67*, *157*
NUMERIC-SEPARATOR attribute
  SESSION handle *157*
-numsep startup parameter *154*
-numdec startup parameter *154*

O

OPEN QUERY statement
COLLATE option *156*
OpenEdge databases
code page of *37*
OpenEdge Explorer *93*
ORACLE DataServer
multi-byte support *108*
OUTPUT TO statement
4GL *34*
OVERLAY statement
multi-byte support *159*

P

Paper sizes
international standards *64*
Parameter files
international *147*
Phone numbers
international conventions *77*
Postal codes
CHARACTER data type *68*
PREFIX function *165*
PRESELECT phrase
COLLATE option *156*
Print Code Page (-cpprint) startup parameter *153*
PrinterFont setting
progress.ini file *64*
Printers
code page of *38*
Printing
paper sizes *64*
setting fonts in progress.ini *145*
Procedure Editor
multi-byte support *108*
PROCOPY utility *171*
PRODB utility *132*, *171*
Progress Developer Studio for OpenEdget
multi-byte support *107*
progress.ini file *64*
PROLIB utility *39*, *171*
PROMSGS environment variable *146*, *152*
PROMSGS file
database server *95*
localized editions *24*
PROSERVE *93*
PROTERMCAP file *34*, *146*
PROUTIL utility
BULKLOAD qualifier *92*
CODEPAGE-COMPILER qualifier *49*
CONVCHAR CHARSCAN qualifier *34*, *89*
CONVCHAR CONVERT qualifier *90*
IDXBUILD qualifier *52*, *58*, *86*
IDXFIX qualifier *58*
PROUTIL utility *(continued)*
WBREAK-COMPILER qualifier *57*
WORD-RULES qualifier *58*
p-snhndl1.p *67*

R

R-code
compiling *107*
R-code In Code Page (-cprcodein) startup parameter *153*
R-code Out Code Page (-cprcodeout) startup parameter *153*
R-INDEX function
multi-byte support *159*
Radix *65*
RCODE-INFO handle
CODEPAGE attribute *157*
LANGUAGES attribute *157*
READKEY statement
multi-byte support *159*
Relational operators *161*
REPEAT function *165*
REPLACE function
multi-byte support *159*
Report Builder
multi-byte support *108*
RIGHT function *165*
RIGHT-TRIM function
multi-byte support *159*
Rounding methods
cultural conventions *77*
RPAD function *166*
RTRIM function *166*

S

Screen layout *74*
Screens
differences *74*
Scripts
specifying
in the progress.ini file *145*
SEARCH function
multi-byte support *159*
Servers
database *93*
SESSION handle *157*
Single-byte characters
definition of *106*
Single-byte code pages
definition of *106*
SmartObjects
instance attributes *80*
Sockets *35*

Sorting
double-byte characters *124–125*
international data *69*
Sounds
selecting *76*
Source code page
definition of *34*
SQL
multi-byte support *108*
Startup parameters
localizable *93*
Storing date data *67*
Stream Code Page (-cpstream) startup parameter *153*
Streams *34*
STRING function
multi-byte support *159*
stty command (UNIX)
-istrip flag *126*
SUBSTITUTE function
multi-byte support *159*
text translation *84*
SUBSTR function *166*
SUBSTRING function
multi-byte support *159*
SUBSTRING statement
multi-byte support *160*
SUFFIX function *166*

T

Table dump (.d) files
code page of *39*
loading *91*
Target code pages
definition of *34*
Terminal Code Page (-cpterm) startup parameter *153*
Terminals
character
code page of *37*
TERMINATOR
word-delimiter attribute *183*
Text editors *108*
Text files
code page of *39*
Text formats
addresses *75*
currency values *76*
dates and times *76*
symbols and abbreviations *77*
Thousands Separator (-numsep) startup parameter *148*, *154*
Time
CHARACTER data type *68*
international standards *68*
Time-format literals *162*
Trail bytes
definition of *106*
sorting *125*
Trailers
code<<#106>page *91*
Translation
labeling strings for *63*
user interface text *83*
Translation Management System
multi-byte support *108*
TRIM function
multi-byte support *160*
Triple-byte characters
definition of *106*
Triple-byte code pages
definition of *106*

U

UCASE function *166*
Undefined code page
definition of *35*
Unicode
advantages of *128*
definition of *128*
designing queries for *133*
fonts for *130*, *134*
normalization *134*
Progress support for *23*
UNIX *34*
UPPER function *166*
USE_IT
word-delimiter attribute *183*
User-defined characters *123*
UTF-8 *48*

W

WBREAK-COMPILER qualifier
PROUTIL utility *171*
Windows screen and keyboard
code-page of *38*
Word-break behavior *125*
Word-break tables
associating with databases *58*
compiling *57*
creating and modifying *54*
providing access to *57*
rationale *53*
syntax *56*, *181*
Word-delimiter attributes *54*
WORDRULES qualifier
PROUTIL utility *171*

Y

YEAR function *67*
-yr4def startup parameter *154*
-yy startup parameter *154*