Building International Applications

Release 3.5 of Forte™ 4GL

Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, CA 94303 U.S.A. 1-800-786-7638

Part No. 806-6663-01
October 2000, Revision A

Contents

2 Building International Applications

3 Specifying Locale Settings

4 Command Line Utilities

Preface

Building International Applications provides information about designing and implementing applications that can be run using multiple languages and national conventions. Topics covered in this manual include:

- building message catalog files
- using Forte 4GL classes to access message catalogs
- running distributed multilingual applications
- using the Extmsg and Compmsg utilities
- using international applications from an end-user perspective

Organization of This Manual

Building International Applications combines task-oriented information for application developers and information for the end users of international applications. This manual contains the following chapters:

Chapter	Description
Chapter 1, "Introduction"	Describes the concepts and terminology used to discuss building and using international applications.
Chapter 2, "Building International Applications"	Provides instructions for building multinational and multilingual applications.
Chapter 3, "Specifying Locale Settings"	Provides information about environment variables and other settings related to using international applications.
Chapter 4, "Command Line Utilities"	Contains information about utilities used to build international applications.

Conventions

This manual uses standard Forte documentation conventions in specifying command syntax and in documenting TOOL code.

Command Syntax Conventions

The specifications of command syntax in this manual use a “brackets and braces” format. The following table describes this format:

Format	Description
bold	Bold text is a reserved word; type the word exactly as shown.
italics	Italicized text is a generic term that represents a set of options or values. Substitute an appropriate clause or value where you see italic text.
UPPERCASE	Uppercase text represents a constant. Type uppercase text exactly as shown.
underline	Underlined text represents a default value.
vertical bars |	Vertical bars indicate a mutually exclusive choice between items. See braces and brackets, below.
braces { }	Braces indicate a required clause. When a list of items separated by vertical bars is enclosed in braces, you must enter one of the items from the list. Do not enter the braces or vertical bars.
brackets [ ]	Brackets indicate an optional clause. When a list of items separated by vertical bars is enclosed by brackets, you can either select one item from the list or ignore the entire clause. Do not enter the brackets or vertical bars.
ellipsis …	The item preceding an ellipsis may be repeated one or more times. When a clause in braces is followed by an ellipsis, you can use the clause one or more times. When a clause in brackets is followed by an ellipsis, you can use the clause zero or more times.

TOOL Code Conventions

Where this manual includes documentation or examples of TOOL code, the TOOL code conventions in the following table are used.

Format	Description
parentheses ( )	Parentheses are used in TOOL code to enclose a parameter list. Always include the parentheses with the parameter list.
comma ,	Commas are used in TOOL code to separate items in a parameter list. Always include the commas in the parameter list.
colon :	Colons are used in TOOL code to separate a name from a type, or to indicate a Forte name in a SQL statement. Always include the colon in the type declaration or statement.
semicolon ;	Semicolons are used in TOOL code to end a TOOL statement. Always type a semicolon at the end of a statement.

The Forte Documentation Set

Forte produces a comprehensive documentation set describing the libraries, languages, workshops, and utilities of the Forte Application Environment. The complete Forte Release 3 documentation set consists of the following manuals in addition to comprehensive online Help.

Forte 4GL

- *A Guide to the Forte 4GL Workshops*
- *Accessing Databases*
- *Building International Applications*
- *Escript and System Agent Reference Manual*
- *Forte 4GL Java Interoperability Guide*
- *Forte 4GL Programming Guide*
- *Forte 4GL System Installation Guide*
- *Forte 4GL System Management Guide*
- *Fscript Reference Manual*
- *Getting Started With Forte 4GL*
- *Integrating with External Systems*
- *Programming with System Agents*
- *TOOL Reference Manual*
- *Using Forte 4GL for OS/390*

Forte Express

- *A Guide to Forte Express*
- *Customizing Forte Express Applications*
- *Forte Express Installation Guide*

Forte WebEnterprise and WebEnterprise Designer

- *A Guide to WebEnterprise*
- *Customizing WebEnterprise Designer Applications*
- *Getting Started with WebEnterprise Designer*
- *WebEnterprise Installation Guide*

Forte Example Programs

In this manual, we often include code fragments to illustrate the use of a feature that is being discussed. If a code fragment has been extracted from a Forte example program, the name of the example program is given after the code fragment. If a major topic is illustrated by a Forte example program, reference will be made to the example program in the text.

These Forte example programs come with the Forte product. They are located in subdirectories under $FORTE_ROOT/install/examples. The files containing the examples have a .pex suffix. You can search for TOOL commands or anything of special interest with operating system commands. The .pex files are text files, so it is safe to edit them, though you should only change private copies of the files.

Viewing and Searching PDF Files

You can view and search 4GL PDF files directly from the documentation CD-ROM, store them locally on your computer, or store them on a server for multiuser network access.

Note You need Acrobat Reader 4.0+ to view and print the files. Acrobat Reader with Search is recommended and is available as a free download from http://www.adobe.com. If you do not use Acrobat Reader with Search, you can only view and print files; you cannot search across the collection of files.

To copy the documentation to a client or server:

1. Copy the fortedoc directory and its contents from the CD-ROM to the client or server hard disk.

   You can specify any convenient location for the fortedoc directory; the location is not dependent on the Forte distribution.

2. Set up a directory structure that keeps the fortedoc.pdf and the 4gl directory in the same relative location.

   The directory structure must be preserved to use the Acrobat search feature.

Note To uninstall the documentation, delete the fortedoc directory.

To view and search the documentation:

1. Open the file fortedoc.pdf, located in the fortedoc directory.
2. Click the **Search** button at the bottom of the page or select **Edit > Search > Query**.
3. Enter the word or text string you are looking for in the Find Results Containing Text field of the Adobe Acrobat Search dialog box, and click **Search**.

   A Search Results window displays the documents that contain the desired text. If more than one document from the collection contains the desired text, they are ranked for relevancy.

Note For details on how to expand or limit a search query using wild-card characters and operators, see the Adobe Acrobat Help.

4. Click the document title with the highest relevance (usually the first one in the list or with a solid-filled icon) to display the document.

   All occurrences of the word or phrase on a page are highlighted.

5. Click the buttons on the Acrobat Reader toolbar or use shortcut keys to navigate through the search results, as shown in the following table:

Toolbar Button	Keyboard Command
Next Highlight	Ctrl+]
Previous Highlight	Ctrl+[
Next Document	Ctrl+Shift+]

6. To return to the fortedoc.pdf file, click the Homepage bookmark at the top of the bookmarks list.
7. To revisit the query results, click the **Results** button at the bottom of the fortedoc.pdf home page or select **Edit > Search > Results**.

Chapter 1

Introduction

An international application can adapt at runtime to different languages and formatting conventions. Forte provides several levels of support for building international applications that include multinational and/or multilingual features. A Forte multinational application, for example, uses information specified by the end user, either through control panels or environment variables, to adapt at runtime to specific conventions for formatting dates, time, numbers, and currency. And to support the development of multilingual applications, Forte supplies a set of translated runtime error messages, predefined locale description files, and special classes and methods.

In addition to a discussion about international concepts, this chapter provides an introduction to international concepts and an overview of Forte 4GL's support for international features.

About International Applications

There are two distinct areas to consider when developing international applications. One area is territory-specific conventions, such as how date, time, money, and numeric fields should be displayed. These conventions vary between geographic regions. An application that can respond to different territory conventions at runtime is called a multinational application. The other area to consider is the language used to display all text visible to an end user—including all error and informational messages and literal strings that appear on the screen. An application that can adapt to a different language at runtime is called multilingual. You may need to consider only one of these areas, or both.

Territory and language conventions are characteristics of a particular user's *locale*. An international application takes its cue from a system's locale settings. Locales, and other concepts involved in internationalization, are described below.

What Is a Locale?

As mentioned above, a locale is a set of information that describes the characteristics specific to a language or territory. For example, users in England have different locales than users in Japan—not only do the users use different languages, but they also use different conventions for formatting currency, numbers, dates, and time. Likewise, U.S. users use a similar locale to British users—both use English—but still represent currency, dates, and time differently.

The following information characterizes a locale:

Collating Sequence The collating sequence describes how a particular locale performs a sort function. For example, compare how a United States English locale and a Spanish locale alphabetize the same list of words:

Original List	US Sort	Spanish Sort
cull	cheque	cult
cult	chess	cull
cheque	cull	czech
czech	cult	cheque
chess	czech	chess

The Spanish convention treats 'ch' and 'll' as multi-character collating symbols that collate after 'c' and 'l', respectively.

Character Classification In general, characters can be classified into alphanumeric, alphabetic, control, digit, graphic, lowercase, printable punctuation, space, and uppercase. An international product must be able to place characters in the proper classifications to ensure proper results from TextData methods, such as IsAlpha(), IsUpper(), and ToUpper(). Some languages have multi-byte character sets, which are described in the section "What Is a Multi-Byte Character?" on page 16.

Currency Format Strings and Symbols Currency format refers to the representation of currency in each locale. For example, the string 12,345.67 is represented in territory formats as shown below:

Locale Currency	Format
US dollar	$12,345.67
Deutschemark	DM 12.345,67
Japanese Yen	¥12,345.67

Numeric, Date, and Time Formats A locale specifies how to format the following:

- symbols used for numeric decimal point, thousands separator

  The symbols used for numeric decimal point thousands separator are locale dependent. For example, the US decimal point is a period (.), and the German decimal point is a comma (,).

- the number of digits grouped between separators

  For example, in the U.S., the grouping number is three (3)—the value 1234567 when grouped becomes 1,234,567.

- date/time format strings, year names, day and month names

  The translation of day, week, and month names, as well as the format of dates and times, are locale dependent. The following table displays example formats of the date and time “Tue, April 2, 1994 1:11:44 pm”:

Locale	Format
US	Tue, April 2, 1994 1:11:44 PM
France	Mar., 02, Avr. 1994, 13:11:44

Locale Files

Forte supplies a set of locale files containing default settings for all the locale information described above, as well as the specification of the current language. The system manager specifies the default locale file using the FORTE_LOCALE environment variable. There are additional environment variables that allow a system manager to override specific settings in the locale file.

In addition, on Windows platforms, the end user of a Forte application can use control panels to specify language choice and formatting conventions. Forte also provides environment variables to supplement the native mechanisms. On UNIX and VMS, the end user specifies language options with a set of environment variables and resource files.

The complete listing of supported locale files, as well as the precedence rules for determining which settings are active, can be found in Chapter 3, “Specifying Locale Settings.”

What Is a Territory?

A territory is a distinct geographic region that has its own recognized standards for formatting numbers, currency, dates, and time. Forte always uses language and territory in combination. For example, Canada, the United States, and England are territories associated with the English language; parts of Canada (for example, Quebec) and France are territories of the French language.

What Is a Codeset?

A codeset is the computer's internal representation of a character set.

Examples of codesets include:

- ASCII
- ShiftJIS
- ISO8859-1

What Is a Multi-Byte Character?

Most character sets contain a small number of characters and can be represented using 8 bit representation (maximum number of characters is 256), with single-byte storage. Some character sets (Japanese, Chinese, Korean) contain a few thousand characters, which require 16, 24, or 32-bit representation. Thus, for large character sets the unit of representation must be 2, 3, or 4 four bytes, or multi-byte representation.

When manipulating multi-byte characters using TextData methods such as MoveTo and SizeNext, an international application must make sure to leave the TextData object in a consistent state, positioned at the start of a multi-byte character.

What Is a Multinational Application?

A multinational application can adapt at runtime to end users' language and territory settings. For example, while British users use English, their conventions for formatting currency, dates, and time are different than users of American English. A multinational application can accommodate different territory conventions.

What Is a Multilingual Application?

A multilingual application is an application that can be run in multiple languages. There are two kinds of multilingual applications:

- an application that responds to language settings and comes up in one of its supported languages

  For example, a software company might provide versions of an application for its French, German, and Japanese distributors. This application will be run in one language only, depending on the country in which the application is used. Typically, the language setting will determine the source of message strings and menus presented to the end user at runtime.

- an application that starts in one of any number of languages, but can change language during runtime in response to an end user's selection

  For example, banks are likely to have to service customers that speak different languages. A useful menu item on a banking application might be a list of languages from which the teller can choose to view the screens. Once the teller selects a language, all subsequent windows will change to reflect the new language and conventions. The teller can print out statements for customers in the appropriate language, and will be able to read the screens in the appropriate language, avoiding ad hoc translation during a banking transaction.

Both types of multilingual applications can use one installed application regardless of the number of languages each application supports.

What Is a Message Catalog?

A message catalog is a file containing all text strings (including menu, button, and field labels) for an application. The application developer creates the catalog during the application development process. Forte provides a utility to help automate the creation of the message catalog—see the section "Extmsg Utility" on page 20 for more details.

The message catalog supplies the text necessary for translation—each supported language requires its own message catalog. An internationalized application will access the appropriate translated text at runtime with invocations of the MsgCatalog methods GetString or GetTextData. See the section "Accessing Message Catalogs" on page 32 for information about accessing message catalogs.

About Forte's International Support

This section provides a brief overview of Forte's support for multinational and multilingual applications. Chapter 2, "Building International Applications," provides implementation details for the application developer. Chapter 3, "Specifying Locale Settings," contains information about controlling international features through control panel settings and environment variables.

International Templates

As mentioned above, some international applications will allow end users to change the language in which all visible text is displayed, as well as the territory conventions applied to the text. All Forte applications using standard format templates will automatically adapt to a language or territory change by translating text, such as days of the week, to the appropriate language. In addition, there are several special formatting templates that you can use to force the application to apply the specific conventions set by the user. For example, the international DATE template will force an application to use the specific formatting instructions provided by the locale file or specified through control panel settings or environment variables. In this way, an application can adapt to changes in territory conventions.

The international templates are discussed in the section "Preparing Multinational Applications" on page 22.

Message Catalogs

At the heart of a multilingual application is one or more message catalogs. A message catalog contains every visible piece of text displayed in an application—including error messages, field labels, window titles, and so on.

Replace text with method invocations

To build a multilingual application, you extract all messages and text strings from a TOOL application and place them in one or more message files. A language translator would then use any standard system utilities and text editors to translate these files to another language. Meanwhile, in every place you removed text, you place invocations of the GetString or GetTextData methods to access the message file(s) and display the appropriate text.

The Forte distribution is structured in such a way that if the current locale file setting is changed to reflect a new language, the application will access the corresponding message catalog. You create and store message catalogs independently of the TOOL source code, allowing an application to run in a variety of languages while using the same TOOL source code.

Details about using message catalogs to build multilingual applications can be found in the section "Building a Message Catalog File" on page 27.

Classes and Methods

Several special classes, methods, and attributes support internationalized applications and can be found in the Framework Library. The classes, methods, and attributes are the following:

ApplicationDesc class Provides access to information about the "current" application in which TOOL code is executing, giving easy access to the message catalogs.

ExtendedFontDescriptor class Objects of the ExtendedFontDescriptor class represent fonts that might vary across codesets and window systems.

LocaleDesc class Describes locale information such as locale name, language name, and territory ID.

MsgCatalog class Represents a compiled message catalog file, which contains a set of numbered messages that an application accesses at runtime. Methods on this class allow an application to retrieve messages from the file at runtime.

NativeLangMgr class Provides various services supporting international functionality in Forte applications, such as explicitly opening a message catalog, and retrieving a partition's locale information.

Collate method This method appears on the TextData class and compares two strings, returning a positive or negative value depending on the collate sequence.

IsHiragana, IsHankana, IsKanji, IsKatakana methods These methods appear on the TextData class and identify Hiragana, Hankana, Kanji, and Katakana characters, respectively.

GetErrorText method This method appears on the ErrorDesc class and returns the message text of an ErrorDesc object.

GetAppMsgCatalog method This method appears on the NativeLangMgr class and explicitly opens a manually-installed message catalog and returns a reference to the created MsgCatalog object.

SetWithParams method This method appears on the ErrorDesc class and sets the message text and catalog information for an error.

SetLocale method This method appears on the TaskHandle class and allows you to change the locale from within an application.

GetLocale method This method appears on the TaskHandle class and allows an application to retrieve the language and territory portions of the locale.

NativeLangMgr attribute This attribute appears on the Partition class and contains a reference to the NativeLangMgr object created for a partition at startup.

LocaleDesc attribute This attribute appears on the TaskHandle class and contains the locale information for the current task.

Note All TextData methods automatically take into account locale-specific needs and multi-byte character sets.

Locale Files

Forte provides a set of pre-defined locale files. The locale files contain default language and territory conventions for a variety of geographical regions. As mentioned earlier, the default locale file is set using the FORTE_LOCALE environment variable. However, users can override default settings with control panels and other environment variables. Refer to Chapter 3, "Specifying Locale Settings," for more information about locales. A complete list of supported locales can be found in the file $FORTE_ROOT/install/nls/locale.

Translated Message Files

To support the use of multilingual applications, Forte supplies a set of translated message files. Applications reference the translated files at runtime according to the language setting specified by current locale file or environment variable. A list of the complete set of translated files can be found in $FORTE_ROOT/install/nls/fortemsg. Note that Forte Express message files are found in $FORTE_ROOT/install/nls/exmsg.

Extmsg Utility

To help convert a TOOL program containing embedded strings, Forte provides the command line utility Extmsg for extracting strings and replacing them with invocations of the GetString method, and placing the extracted strings into message files. The Extmsg utility is discussed in Chapter 4, "Command Line Utilities."

The Extmsg utility is useful to run on existing applications. For new applications, you may find it more efficient as you develop the application to manually place the text into a message file and include the appropriate method invocations in your code. Or, you can develop a new application in your primary language, and then run the Extmsg utility on it.

Compmsg Utility

Once translators translate the message file, you then run it through a command line utility, called Compmsg, to build the compiled and portable version of the file, called the message catalog. The message catalog is an indexed file that is portable across machines and different codesets.

The translation step can be done either before or after a distribution is made for a TOOL application. The message file is simply represented in a distribution as a source and compiled version. As a result, the translations can be done at the deployment site, or during development. The Compmsg utility is discussed in the section Chapter 4, "Command Line Utilities."

Codeset Conversion

Forte applications are distributed and may run across a wide variety of machine architectures. Because different machines can have different codesets, it is likely that a distributed application will be required to pass textual data across different codesets. Forte handles this situation by automatically converting codeset-dependent text data when passing such data between dissimilar codesets.

Where to Go from Here

If your application will be deployed in several languages, but only within a single country, you will mainly need to concern yourself with the language sections of this chapter. See "Building Multilingual Applications" on page 25.

If your application will be deployed in several countries, but always in the same language, you will need to pay more attention to the section about territory considerations. See "Preparing Multinational Applications" on page 22 and Chapter 3, "Specifying Locale Settings."

If you have a multilingual, multinational application, read through the entire chapter.

Chapter 2

Building International Applications

As mentioned in Chapter 1, "Introduction", Forte 4GL provides a set of features for implementing international applications. There are several Forte features that support an application's adaptation to territory conventions, as well as several special features for implementing multilingual applications. This chapter covers the following topics:

- international formatting templates
- using multi-byte characters
- building multilingual applications
- translating windows
- using extended fonts
- running distributed applications
- accessing external resources

Preparing Multinational Applications

The features described below support the development of applications that will be run in multiple territories. Use these features in conjunction with the features that support multilingual applications to prepare multinational, multilingual applications.

Formatting Numbers, Currency, Dates, and Time

It is likely that you have used, or will use, Forte templates to format numbers, currency, dates, and time. As mentioned in Chapter 1, "Introduction," international applications rely on locale files and/or locale environment variable settings to determine formatting conventions. The behavior of any templates you implement in your applications is dependent on the settings in the current locale, or any environment variables that might override locale file settings. The following sections describe how each set of templates acclimates to the current locale.

Formatting Numbers

The NumericFormat class provides templates that format numeric data, including integer, floating point, and decimal (money) data. Refer to the Forte online Help for a complete listing of NumericFormat templates.

At runtime, Forte numeric formats adapt to locale specifications as follows:

- The decimal point in numeric formats is replaced by the radix character specified in the locale preference.
- The comma in numeric formats is replaced by the thousands separator character specified in the locale preference.
- The grouping convention is implemented as specified in the locale preference.

U.S. English is default

If locale information is missing for something specified in the format templates, Forte uses U.S. English values.

Formatting Currency

Use the special CURRENCY template to request that the local currency formatting conventions be used to format decimal data. At runtime, any data formatted with the CURRENCY template will conform to the monetary specification in the current locale file. For example:

```
number : DoubleData = new(value=123.45);
numfmt : NumericFormat = new;
numberAsText : TextData;
numfmt.Template = TextData(value='CURRENCY');
numberAsText = numfmt.FormatNumeric(source = number);
...
number = numfmt.DecodeDouble(source = numberAsText);
```

In some languages, currency is formatted differently than numbers. Forte provides a special convention for formatting currency, which involves prefixing any numeric format that deals with currency with an exclamation point (!). The '!' tells Forte that it will be dealing with currency (rather than numbers), and thus Forte will use the locale preferences for the separator character, grouping, and the decimal point. Refer to the Forte online Help for more information on numeric formats.

Note If you use the standard formats, the currency symbol (the American $) is not replaced in all formats by the locale conventions. For example, if you use the format "$,$$$.99", Forte will display the data as "$1,234.56"—even if the application is run in a territory that uses DM as the currency symbol. Therefore, to ensure proper conventions, use the CURRENCY format.

Formatting Dates and Times

Forte provides international and standard templates in the DateFormat class to format date and time data. Refer to the Forte online Help for a complete listing of DateFormat templates. Both types of templates behave according to locale specifications, as are described below.

International Templates

Forte provides date and time templates to request that an application use territory-specific formatting conventions at runtime. The formats are specified in the locale file, or through control panels or environment variables. You apply these templates as you do regular date and time templates (within TOOL code or in the Window Workshop). The following international templates are available:

Template	Description
DATETIME	Specifies that Forte use the local date/time formatting conventions to format the date and time in a short format.
DATE	Specifies that Forte use the local date formatting conventions to format the date in a short format.
LONGDATETIME	Specifies that Forte use the local date/time formatting conventions to format the date and time in a long format.
LONGDATE	Specifies that Forte use the local date formatting conventions to format the date in a long format.
TIME	Specifies that Forte use the local time formatting conventions to format the time.
aaa	Displays the abbreviated name of the day of the week in English, regardless of the current locale setting.
aaaa	Displays the full name of the week in English, regardless of the current locale setting.
nnn	Displays the name of the month in English, regardless of the current locale setting.
nnnn	Displays the full name of the month in English, regardless of the current locale setting.
g	Displays the Japanese era name as the first character in the Romanji representation of the era name.
gg	Displays the abbreviated Japanese Era name as a single Kanji character.
ggg	Displays the full Japanese Era name as a double Kanji character.
e	Displays the year within the Japanese Emperor era as a number without a leading zero.
ee	Displays the year within the Japanese Emperor era as a number with a leading zero.

The following example shows how the international DATE template uses the current locale settings. Assume that the end user has set the date format in the control panel of a PC to include Weekday, Date, Month, and Year, using no separators—the language is French and the territory is Canada:

Apply the DATE template

```
dt : DateTimeData = new;
dtfmt : DateFormat = new;
DateAsText : TextData;
dt.SetCurrent;
dtfmt.Template = 'DATE';
DateAsText = dtfmt.formatDate(source=dt);
```

The date will look like the following: Samedi 21 mars 1992. Compare this example with the next example, which uses a standard template.

Standard Templates

At runtime, standard Forte date and time formats adapt to locale specifications as follows:

- The month names and abbreviations (specified by the 'mmm' and 'mmmm' templates) are replaced by the language equivalents specified in the locale preference.
- The AM/PM designations (specified by the 'AM/PM', 'am/pm', 'A/P', and 'a/p' templates) are replaced by the language equivalents specified in the locale preference.
- The weekday names and abbreviations (specified by the 'ddd' and 'dddd' templates) are replaced by the language equivalents specified in the locale preference.

The following example uses a standard format along with the locale specification described in the previous example:

Apply a standard template

```
dt : DateTimeData = new;
dtfmt : DateFormat = new;
DateAsText : TextData;
dt.setCurrent;
dtfmt.Template = 'mmm dd,yy';
DateAsText = dtfmt.formatDate(source=dt);
```

If you run this code with the same settings described above (French Canadian), the date will look like the following: Mars 21, 1992.

Using Multi-byte Character Formats

The character format templates in the TextFormat class provide a format template ('k') that supports the multi-byte Kanji characters. However, the standard character template '?#' will accept one Kanji character input (which may be more than one byte). Kanji is the only language supported by Forte that requires a special character format template. Refer to the Forte online Help for a complete listing of TextFormat templates.

Writing Portable Code

The ASCII codeset ("asc") defines only 127 characters, which correspond to the upper and lower case Roman characters (without accent), all of the digits, and a number of special characters (punctuation, arithmetic symbols, and so on). This codeset is usually sufficient for standard UK and American applications.

All of the defined codesets used by Forte contain the identical ASCII subset of characters as the first 127 one-byte characters. As a result, any partition that starts up defined with the ASCII codeset will not need to convert between codesets, because all codesets use the ASCII codes. However, if you use the ASCII codeset designation and actually send characters with codes that are not in the ASCII character set, no conversion will occur, and you may get corrupted information. Refer to the section "Partitioned International Applications" on page 47 for more information about codeset conversion.

To ensure maximum portability of code across codesets, TOOL identifier names must consist only of a portable set of ASCII characters. These identifiers include class names, project names, local variables, and so on. The Forte workshops enforce the use of this set of characters. Note that the contents of strings, and the names of database tables and columns do not need to be restricted to ASCII. Refer to the *TOOL Reference Manual* for information about Forte naming.

Building Multilingual Applications

The basis for building multilingual applications is the use of method invocations to access a message catalog. A message catalog is an indexed file, external to the TOOL program, which contains all the text that appears in an application—this includes error messages, status lines, field labels, and menus. Each message in the message catalog is uniquely numbered and corresponds to an appropriate method invocation. The message catalog is translated into each language your application will support.

By providing a set of files with different language translations of the messages, but with the same numeric identifiers, a single TOOL program can access the appropriate language at runtime to adapt to the user's language preferences.

Summary of Development Process

The general steps for developing a multilingual application are the following:

1. Construct and test the application in the "primary" language intended for its use.
2. Export the projects that comprise the application to .pex export files.
3. Use the Extmsg utility to make a first cut at converting strings into messages. (See Chapter 4, "Command Line Utilities," for more information.)

   The Extmsg utility simply takes strings out of an export file and replaces them with invocations of the GetString and GetTextData methods. The utility places the strings into a file that you can use as a basis for the message catalog, or you can add the file to an existing message file for the application. (See "Creating a Message Source File" on page 27.)

   Strings that you do not want extracted should be bracketed by the function NX().(NX stands for No Translation.) For example:

   ```
   username.Value = NX('?');
   ```

   Note that you can initially develop your application using the GetString and GetTextData methods and manually build the message file rather than putting in the text and then extracting it. This allows better control over the numeric identifiers assigned to messages.
4. Re-import the projects into the repository.
5. Within the development environment, check the TOOL code and edit or remove any remaining hard-coded messages.
6. Add more method invocations if necessary, and add the referenced messages to the text message file. You may want to edit the message file to remove some extracted messages and put them back into the TOOL source code. (See "Accessing Message Catalogs" on page 32).
7. Use the widget property dialogs to specify message numbers and message set numbers for all static text that appears on windows. Add the corresponding text to the message catalog.
8. Put the message file created for the application into the FORTE_ROOT/workmsg language subdirectories on the machine upon which you are developing the application. (See "Location of Message Catalogs" on page 31.)

   Note that all message files are external to the repository, and it is the developer's responsibility to place these files in the proper location.

9 Use the Compmsg utility to compile the message file into portable binary format. You must run the Compmsg utility any time you add, modify, or delete messages in the message file. (See "Compiling a Message Source File" on page 30).

The steps above are described in the sections that follow. Please note that this chapter assumes that you have an existing application that you wish to internationalize. Therefore, the steps include running the Extmsg utility to create a message source file. Of course, you can also build this file manually.

Building a Message Catalog File

As mentioned earlier, a message source file contains text messages identified by number, in ascending (but not necessarily consecutive) order. You generate a message catalog from a message source file—a message catalog file is the compiled version of the message source file. Applications access messages in the catalog by invoking the GetString or GetTextData methods to read in messages from the catalogs.

Creating a Message Source File

You create a message source file in one of two ways: by running the Extmsg utility on the .pex files of an existing application, or by manually creating a file (using the text editor of your choice) and entering the message as you develop your application.

Extmsg utility

If you use the Extmsg utility, the utility automatically replaces the text strings with invocations of the GetString method (if you want the GetTextData method, you have to put it in manually). Refer to Chapter 4, "Command Line Utilities," for information about the Extmsg utility.

If you create the file manually, you must be sure that you also modify your TOOL code to include the appropriate GetString or GetTextData invocations. Using these methods is described in the section "Accessing Message Text" on page 33.

Source file name

You must name the message source file the same name as the unique application identifier (first 8 characters of the name), plus a .msg suffix. For example, if your application is called myapp, the message source file would be myapp.msg.

Regardless of how you choose to create the message source file, you can expect to modify the file throughout the development of your application. The following section describes the formatting conventions of a message source file.

Formatting a Message Source File

Each message in a source file begins with a number, followed by a single space, followed by text, and terminated by a newline character. Message numbers are unsigned integers and must be in ascending order—numbers need not be consecutive.

The sections that follow describe various formatting semantics for preparing message source files. An example of a message source file is shown below. Forte uses the X/Open format conventions for language-dependent message files.

A message text source file looks like the following example:

Comment

Escape character

$quote "

```
$ Here are the messages for the errors in my app
1 Error entering %1 workshop by user %2.
3 You must enter the following: \n\
 1. your name.\n\
 2. your password.
$quote "
7 "  here is some text with leading blanks"
8 "here is some text with trailing blanks    "
$quote
11 This is back to specifying without quotes
```

Source File Comments

Any line beginning with a dollar sign ($) followed by a single space or tab is considered a comment. You can use comment lines to put in descriptive information that might be handy during the translation process.

Using the Backslash Escape Character

You use the backslash (\) character in a message as an escape character in the following ways:

Continuing Lines A backslash followed immediately by a newline allows you to enter a long message across several lines. For example:

```
7 This message text spans several lines. In this file, the \
message is on 4 lines, but when displayed, it is actually one \
long line. The \ character at the end of each line denotes \
continuation.
```

Inserting a Newline A backslash followed by a newline character (\n) inserts a newline—a carriage return is implied. For example:

```
8 This message will be displayed\n on two lines.
```

Inserting a Tab A backslash followed by a tab character (\t) inserts a tab. For example:

```
10 First\tName:
```

At runtime, the above example will be displayed like the following:

```
First Name:
```

Converting Hexadecimal Numbers A backslash followed by an ‘x’ and a hexadecimal number (\xNN) converts the hexadecimal number into the character whose binary value in the current codeset matches the hexadecimal value. The valid values for the hexadecimal number are in the range 00-FF, and are codeset dependent. This conversion feature is useful for entering multi-byte characters or European characters where the keyboard mapping may not be obvious. For example, the following message text format will convert the hex number to the ‘é’ character at runtime, using the Window Latin 1 codeset:

```
11 Fort\xe9
```

Another example:

```
12 \x82\xa0
```

When displayed, message 12 (in the Japanese SHIFT-JIS codeset) appears as the first vowel (pronounced ‘ah’) of the Japanese Hiragana character set.

Converting Octal Numbers A backslash followed by an octal number (\NNN) converts the octal number represented by 'NNN' to the binary value that is exactly NNN (in the current codeset). Valid values are in the range 000-377, and the values are codeset dependent. This conversion feature is useful for entering multi-byte characters or European characters where the keyboard mapping may not be obvious. For example, the following message text format will convert the octal number 351 to the 'é' character at runtime (using the Windows Latin 1 codeset):

```
13 Fort\351
```

Another example:

```
14 \202\240
```

When displayed, message 14 (in the Japanese SHIFT-JIS codeset) appears as the first vowel (pronounced 'ah') of the Japanese Hiragana character set.

Leading/Trailing Spaces

To make leading or trailing spaces visible, use the $quote directive followed by a character (such as *, “, #, or @, and so on). Then use the character to delimit the start and end of the text for the message to which you wish to add spaces. When you use the $quote directive to specify a character, you must begin and end the text that follows with that character.

To reset the $quote directive, place the $quote directive on a line by itself, with no delimiting character, as shown below.

$quote “

```
$quote “
7 “  here is some text with leading blanks”
8 “here is some text with trailing blanks    “
$quote
11 This is back to specifying without quotes
```

Parameter Placeholders

To pass a parameter into message text, you must leave a placeholder for the parameter in the message source text. The placeholder for the message source text is the percent sign (%). For example:

```
1 Error entering %1 workshop by user %2.
```

To insert the parameter value, use the ReplaceParameters method on a TextData object, which is described in the section “Passing Parameters into Message Text” on page 35.

Grouping Related Sets of Messages

For large numbers of messages or related groups of messages, it is useful to group messages into message sets. Message sets provide a way to subcategorize message numbers and to keep related messages together in a file. In addition, Forte can read in and cache a message set at runtime using the LoadSet/UnloadSet methods on the MsgCatalog class, providing more efficient access to a set of messages. Refer to "Accessing Message Sets" on page 35 for more information about the LoadSet and UnloadSet methods.

$set

You identify a message set by placing a $set directive in the message source file prior to the first message in the set. All messages that follow belong to the set specified by the most recent $set directive. Like message numbers, set numbers are unsigned integers and must be in ascending order, but need not be consecutive. The same message numbers can appear in different sets. If the $set directive does not appear in the message text source file, the set number for all messages is assumed to be 1.

Note

Message set numbers above 60,000 are reserved for Forte.

A message source text file that uses message sets looks like the following:

Example: message sets

```
$set 1
$ Here are the messages for the errors in my app
1 Error entering %1 workshop by user %2.
2 You must enter the following: \n\
 1. your name.\n\
 2. your password.
$set 2
2 The first message in the second set.
4 The second message, even though I skipped a number.
```

Compiling a Message Source File

After you have completed building the message source file, you compile it into a message catalog file by using the Compmsg utility (see Chapter 4, "Command Line Utilities,"). This utility takes a message source file and outputs a file that is portable across operating systems and codesets. An example use of this utility is shown below:

```
compmsg -m myapp.msg -c myapp.cat
```

Forte accesses the myapp.cat file at runtime.

Location of Message Catalogs

To facilitate access to and maintenance of message catalogs, create a subdirectory for each language your application will support in the FORTE_ROOT/workmsg directory. You can either specify a single directory for a language or for a language/territory combination. The name of each directory must correspond exactly to an official Forte language or language/territory name, such as en, en_us, or fr, fr_fr. You can always add subdirectories to the deployment distribution after the application has been deployed. Refer to the $FORTE_ROOT/install/nls/locale for a complete list of supported locales.

For example, if you are distributing an application to several Spanish speaking countries, you can supply one /es directory. All applications that use a Spanish locale—regardless of any territory specification—will use the message catalog in that directory. In addition, Forte gives you the flexibility to also include a territory-specific directory, which will be used if that locale is defined.

You can also create both types of directories, in which case the more general language-only directory will be used when the specific territory directory does not exist.

Once you have created this directory structure, created message source files and then compiled them into message catalogs, place both the .msg and .cat files for each language in the appropriate language subdirectory. For example:

Standard directories with language subdirectories

```
FORTE_ROOT
  /workmsg
    /en
      <appID>.msg
      <appID>.cat
    /en_us
      <appID>.msg
      <appID>.cat
    /fr
      <appID>.msg
      <appID>.cat
```

The directory structure described above ensures an application's access to the proper language files at runtime. In addition, if you partition and test a distributed application, Forte will send the appropriate message catalog files to the nodes on which the test run of any distributed partitions are to be run. This make it very convenient to finalize and test with different partitioning schemes. You can even test the multilingual nature of your application from the Forte Workshops by starting up other clients (with the TestClient utility) that use different language and territory codes. See the *Forte 4GL Programming Guide* for information on the TestClient utility.

Multi-User Application Development

To allow multiple developers to work on the same application with different message catalogs, Forte provides the FORTE_WORKMSG environment variable. FORTE_WORKMSG allows you to specify a directory other than FORTE_ROOT/workmsg to house your message catalogs during development. For example:

```
FORTE_WORKMSG = myDirectory/myTests
```

Set up the specified directory (/myTests in the example above), just as you would FORTE_ROOT/workmsg. In other words, create your locale directories, and place the message source and message catalogs under the appropriate directory.

Accessing Message Catalogs

To incorporate a message catalog into a running application, you modify the TOOL code in your application to call the message catalog at runtime. The actual call to the catalog is made by the GetString and GetTextData methods, which reference the specified message text.

If you ran the Extmsg utility on an application, the utility inserted GetString invocations in every place it extracted a string (Extmsg uses only GetString). If you are manually creating your message source file, then you need to insert these GetString or GetTextData calls yourself. The following sections describe how to use the GetString and GetTextData methods on the MsgCatalog class to access a message catalog at runtime.

Using the MsgCatalog Class

Before an application can make use of a message in a specific message catalog, the application must open the right catalog. An application determines which message catalog to use by the current locale settings. If you have set up your files according to the instructions in the section "Location of Message Catalogs" on page 31, your application can implicitly open the appropriate message catalog before the first GetString method retrieves a message string.

After an application implicitly opens a message catalog, it retrieves messages using the GetString or GetTextData methods on the MsgCatalog class. Both methods have parameters to identify the set number and message number within the set to retrieve.

Referencing a Message Catalog

Forte provides a special TOOL keyword called **application**, which refers to the current ApplicationDesc object for the code (much like the keywords **task** and **transaction**). The ApplicationDesc object has an attribute called **MsgCatalog**, which refers to the message catalog for the current application, while the current locale specifies the required language for the task. You can simply refer to this MsgCatalog object to access the messages for the current application, without having to pass around a MsgCatalog object as a parameter to all of your methods.

The following code shows how to reference the catalog of the "current" application:

```
msg : msgCatalog;
msg = application.MsgCatalog;
```

Message catalogs that are referenced through the ApplicationDesc object are automatically managed by the Forte runtime systems. Forte opens and closes the catalogs based on the message access load. The implicit message catalog is called "<appID>.cat". You can derive this message catalog from any number of source message files by concatenating them before running the Compmsg utility. For example, a common technique is to create a single message source file for each project. That source file contains a set of set numbers that are unique across projects. The source files can be concatenated when you build the catalog for a given application. Using this technique, you can keep just one copy of the messages for a given project, even if that project was used in more than one application.

Explicitly Opening a Message Catalog

Under certain circumstances, you may wish to explicitly open a message catalog. You can open a specific message catalog by invoking the OpenMsgCatalog method on the task.Part.NativeLangMgr object, giving an unopened File object that designates the location of the compiled message catalog file. The OpenMsgCatalog method returns a

MsgCatalog object, which can then be used to access messages in that catalog (see the Forte online Help for more information about these classes and methods). The following code fragment demonstrates:

```
mcat : MsgCatalog;
file_for_cat : File = new;
file_for_cat.SetPortableName
 (name = “/mydisk/locale/french/myApp.cat”);
mcat = task.Part.NativeLangMgr.OpenMsgCatalog
 (file = file_for_cat);
```

This approach, however, is less convenient because you might have to pass the MsgCatalog object reference as a parameter to any methods that might wish to access the messages in that catalog. You also will need to determine the language that is needed, and come up with conventions for identifying the name of the files, and so on. Furthermore, catalogs explicitly opened in this manner are not managed by Forte, and you will have to maintain the file resources associated with the catalog by closing the catalog when not in use.

Explicitly Opening a Managed, Application-specific Message Catalog

The task.Part.NativeLangMgr.GetAppMsgCatalog method opens the message catalog that is installed in $FORTE_ROOT/install/nls/*appname*/*language_territory*.cat. The returned message catalog object is “managed” in the sense that you need not explicitly name the catalog file or close it. Rather, the NativeLangMgr constructs the catalog file name based on the running task’s LocaleDesc and closes the catalog when the partition shuts down. This approach is especially useful in circumstances where a set of common messages used by related applications can be consolidated under one common message catalog instead of being duplicated into multiple catalogs under $FORTE_ROOT/userapp.

The following code fragment shows an example of the GetAppMsgCatalog method:

```
mcat : MsgCatalog =
task.Part.NativeLangMgr.GetAppMsgCatalog('myBaseAp');
mcat.GetString(setNumber=1, msgNumber=3);
```

The NativeLangMgr will attempt to open the message catalog $FORTE_ROOT/install/nls/myBaseApp/en_us.cat by default or if the locale of the running task is en_US.

See the Forte online Help for information about the NativeLangMgr class and the GetAppMsgCatalog method.

Accessing Message Text

The GetString and GetTextData methods on the MsgCatalog class return a string or a copy of a TextData object, respectively, of a message in a message catalog. Both of these methods have two required parameters: setNumber and msgNumber.

setNumber and msgNumber parameters

The setNumber parameter indicates the message set that contains the message, and the msgNumber parameter indicates the specific message string to retrieve.

Note that in most cases, the GetString method is the simpler way to retrieve the message text. Use the GetTextData method if your application will further manipulate the retrieved text, as when you set the value of a menu command.

To access message text:

1 Create a MsgCatalog object and open the current message catalog.

```
mcat : MsgCatalog;
mcat = application.MsgCatalog; -- See below
```

2 Retrieve a text message into a string variable using the GetString method.

```
str : string;
str = mcat.GetString(setNumber=1, msgNumber=3);
```

You can similarly retrieve the message into a TextData object with the GetTextData method:

```
txt : TextData;
txt = mcat.GetTextData(setNumber=2, msgNumber=1);
```

Supplying Default Message Text

You can provide a default message that your application displays when it cannot find a specified message. You might choose to specify the defaultString in the developers' language with an extra indication that the message cannot be found. Use the defaultString parameter with either the GetString or GetTextData method, as shown below:

```
task.GetString(setNumber=2, msgNumber=5,
  defaultString='(default) Invalid employee ID entered.');
```

-l flag with Extmsg

If you use the Extmsg utility to create your GetString or GetTextData method invocations, use the -l option to supply the extracted message to the defaultString parameter for each method invocation.

If you do not provide a default parameter on the GetString or the GetTextData method invocations, and you do not use the -l option with the Extmsg utility, Forte returns a message like the following:

```
ERROR: Set:<number> Msg:<number> does not exist.
```

Accessing Message Sets

If your TOOL code accesses a number of messages from the same set, you can use the LoadSet and UnloadSet methods on the MsgCatalog class to read in and cache a message set at runtime. This technique provides very efficient access to multiple messages. If you use the LoadSet method, you must explicitly unload the set from memory, using the UnloadSet method, to free up memory resources.

To access a message set:

1. Invoke the LoadSet command on the message catalog, specifying the set number you wish to load.

   For example:

   ```
   mcat : MsgCatalog = application.MsgCatalog;
   mcat.LoadSet(setNumber = 3);
   ```

2. Proceed to access messages as usual (see the previous section).

3. Unload the message set by invoking the UnloadSet method.

   For example:

   ```
   mcat.UnloadSet(setNumber = 3);
   ```

Passing Parameters into Message Text

If you have parameters in your message text, you need to establish a TextData object on which to invoke the ReplaceParameters method. The example below shows how parameters are used in the InternatBank example application. Refer to the Forte online Help for details about the ReplaceParameters method on the TextData class.

```
when <WithdrawButton>.Click do
  myMsg : TextData;
  myMsg = mcat.GetTextData(setNumber = WINDOWSET, msgNumber = 9,
   defaultString = NX('Branch %1 out of money today: %2.'));

  myText : TextData = new;
  myText.ReplaceParameters(source = myMsg,
   parameter1 = IntegerData(value = 33),
   parameter2 = <DateField>.TextValue);
  self.Window.MessageDialog(messageText = myText);
```

See InternatBank example **Project**: InternatBank • **Class**: AccountWindow • **Method:** Display

Building Multilingual Windows

All text within windows and widgets displayed in a user interface can be set through attributes and methods described in the Display Library online Help. Using Forte's international features, you can write code to set these attributes from messages read from a message catalog before the window is first displayed. If you design the windows using grid fields and other techniques for letting the interface adapt to the displayed contents, you can achieve excellent portability of interfaces across languages. See the *Forte 4GL Programming Guide* for information about building a portable user interface. Note that you cannot change the base layout of a window using this technique.

Specifying a Message Set for a Window

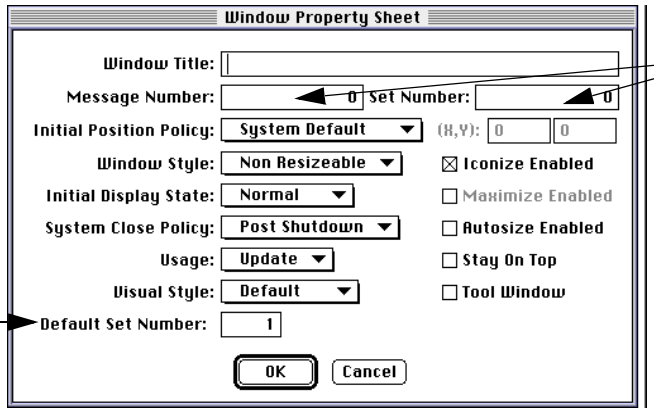
All messages for the widgets on a window can be loaded from the same message set, which is specified as an attribute on a Window object or in a window widget's Properties dialog. Alternatively, you can specify a message set number for each widget (or a group of widgets) on a window; the widget's message set number will override the window's message set number.

On the Window Properties dialog, there are also message number and set number properties for the window's title. If you do not specify a set number for the title, Forte will use the default set number specified for the window. If you do specify a set number for the title, that set will be used.

To specify message and set numbers for a window in the Window Workshop:

1. Enter the Window Workshop.
2. Choose the **File > Window Properties** command.

   The Window Properties dialog is shown in the following figure:



3. Enter a default set number.

   This message set will be used for all the widgets on the window that do not specify their own message set.

4. (Optional) Enter a message number and set number for the window title if you wish to internationalize that text. Remember to enter the message number and corresponding text in the message catalog.

Specifying Message and Set Numbers for Widgets

All widgets that display static text have message numbers as attributes and as properties in the Window Workshop. Message number attributes are not available on widgets that display text only at runtime, such as a data field. You can specify message and set numbers for a widget in your TOOL code or in the Window Workshop. You typically specify message and set numbers in the Window Workshop, unless you are writing a dynamic application that you wish to internationalize. In this case, you can specify the numbers in your TOOL code. Note that regardless of how you specify the numbers, you keep track of them and enter them manually in the message catalog.

Window Workshop property dialogs

For each widget that has a single text value that you can internationalize, there is a message number and set number property. For lists (radio list, drop list, scroll list, fillin field, menu list), there is a Text Value Set Number that specifies the set number for all the list elements, as well as message numbers for individual list elements.

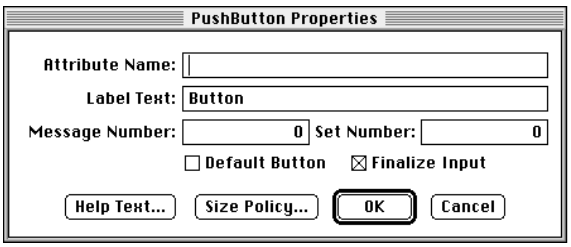
For outline fields and listview fields, there is a Column Title Set Number that specifies the set number for all the column titles.

Default set number

If you do not specify a value for the widget's set number, Forte will use the message set number specified on the widget's window. If you do specify a set number for the individual widget, that set will be used.

To specify message and set numbers for widgets in the Window Workshop:

1. Create a widget in the Window Workshop.
2. Double-click on the widget to display its Properties dialog, or select the widget and choose the **Widget > Properties** command.



3. Enter a message number in the Message Number field and a set number, if necessary, in the Set Number field.

   Remember that you can use the set number for the window as the default set number for every widget on the window. In this case, you do not need to enter a set number in the widget's property dialog, but you must enter a set number in the window's property dialog. However, if you do enter a set number in the widget's Properties dialog, it will override the set number in the window's Properties dialog.

4. In the message catalog, enter the message number and the appropriate text according to the syntax described in "Formatting a Message Source File" on page 27.

Multilingual Help Text for Widgets

All widgets have a Help Properties dialog in the Window Workshop, shown in Figure 1, where you can enter floatover help text and status line help text. Each dialog has a Message Number and Set Number property in which you specify the message number and, optionally, the set number for the status line text. If you do not specify a set number, the help text will use the set number specified on the widget's window.

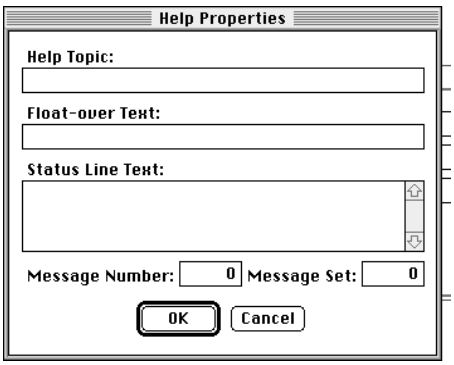


Figure 1 *Help Properties Dialog*

Note There is only one message and (optional) set number for the two types of help—floatover and status line. When you enter the help text in the message catalog, you must use the syntax described in the Display Library online Help, which is the following:

float_over_help **\n** *status_line_help*

This syntax places both help strings on one line in the catalog, and thus only one message number is required.

You can also specify message and set numbers using the following attributes: FloatOverTextMsgNum and FloatOverTextSetNum. These attributes are available on every widget.

Multilingual Help Text for Menu Widgets

Menu widgets also have status line help text that you specify using the **Item > Status Text...** command in the Menu Workshop. The **Item > Status Text...** command displays the Status Text dialog, shown in Figure 2, in which you enter the status line text and its message number and optional set number.

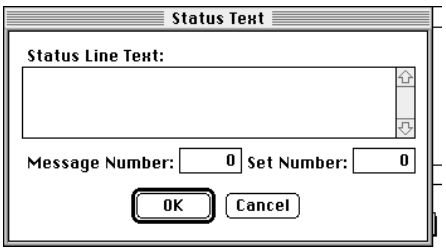


Figure 2 *Status Text Dialog*

You can also specify message and set numbers using the following attributes in your TOOL code: StatusTextMsgNum and Status TextSetNum.

Specifying Message Numbers Dynamically

You can write TOOL code to dynamically display text on a widget according to different application states that you specify in your TOOL code, as shown in the following example:

```
if state == 1
  pb.label = 'Push Me':
else
  pb.label = 'Pull Me';
end if
```

To accommodate a multinational, dynamic application that makes use of message catalogs, you would alter the code to use message numbers and the ReloadLabelText method, code like the following:

```
if state == 1
  pb.labelMsgNum = 45;
else
  pb.labelMsgNum = 46;
end if

pb.ReloadLabelText( );
```

For information about the ReloadLabelText method, see the Forte online help.

If you wish to set the message and set numbers dynamically, the following table contains the widgets and their corresponding attributes.

Widget	Attribute	Description
Window	SetNum	Specifies the default set number for the window. If a widget on the window specifies a set number of zero (the default), then the window's set number is used.
	TitleMsgNum	Specifies the message number for the window's title text. A value of 0 (the default) means that the current value of the Title attribute for the window is used.
	TitleSetNum	Specifies the set number for the message number that contains window's title text.
PushButton	LabelMsgNum	Specifies the message number for the push button's label text. A value of 0 means that the current value of the Label attribute is displayed on the widget.
	LabelSetNum	Specifies the set number for the message number that contains the push button's label text. If you specify a value of 0 (the default), then the window's SetNum value, if any, is used.
ToggleField	TextMsgNum	Specifies the message number for the toggle's text. A value of 0 (the default) means that the current value of the Text attribute is displayed on the widget.
	TextSetNum	Specifies the set number for the message number that contains the toggle's text. If you specify a value of 0 (the default), then the window's SetNum value, if any, is used.
MenuWidget	TextMsgNum	Specifies the message number for the menu widget's text. A value of 0 (the default) means that the current value of the Text attribute is displayed on the widget.
	TextSetNum	Specifies the set number for the message number that contains the menu widget's text. If you specify a value of 0 (the default), then the window's SetNum value, if any, is used.

Widget	Attribute	Description
	StatusTextMsgNum	Specifies the message number for the menu widget's status text. A value of 0 (the default) means that the current value of the StatusText attribute is displayed on the widget.
	StatusTextSetNum	Specifies the set number for the message number that contains the menu widget's status text. If you specify a value of 0 (the default), then the window's SetNum value, if any, is used.
TextGraphic	TextMsgNum	Specifies the message number for the text graphic's text. A value of 0 (the default) means that the current value of the Text attribute is displayed in the status line.
	TextSetNum	Specifies the set number for the message number that contains the text graphic's text. If you specify a value of 0 (the default), then the window's SetNum value, if any, is used.
RadioList	CaptionMsgNum	Specifies the message number for the radio list's caption text. A value of 0 (the default) means that the current value of the Caption attribute is displayed on the widget.
	CaptionSetNum	Specifies the set number for the message number that contains the radio list's caption text. If you specify a value of 0 (the default), then the window's SetNum value, if any, is used.
CompoundField	CaptionMsgNum	Specifies the message number for the compound field widget's caption text. A value of 0 (the default) means that the current value of the Caption attribute is displayed on the widget.
	CaptionSetNum	Specifies the set number for the message number that contains the compound fields widget's caption text. If you specify a value of 0 (the default), then the window's SetNum value, if any, is used.
FieldWidget	FloatOverTextMsgNum	Specifies the message number for the field widget's floatover help text. A value of 0 (the default) means that the current value of the FloatOverText attribute is displayed for the widget.
	FloatOverTextSetNum	Specifies the set number for the message number that contains the field widget's floatover help text. If you specify a value of 0 (the default), then the window's SetNum value, if any, is used.
ListElement	TextValueMsgNum	A value of 0 means that the text entered in the Window Workshop is displayed on the widget.
ListField	TextValueSetNum	Specifies the set number for the message number that contains the list field's text. If you specify a value of 0 (the default), then the window's SetNum value, if any, is used.
FillinField	TextValueSetNum	Specifies the set number for the message number that contains the fillin field's text. If you specify a value of 0 (the default), then the window's SetNum value, if any, is used.
MenuList	TextValueSetNum	Specifies the set number for the message number that contains the menu list's text. If you specify a value of 0 (the default), then the window's SetNum value, if any, is used.
OutlineColumnDesc	TitleMsgNum	Specifies the message number for the column title. A value of 0 means that the current value for the Title attribute is displayed on the widget.
TreeField	TitleSetNum	Specifies the set number for the message number that contains the tree field's column title.If you specify a value of 0 (the default), then the window's SetNum value, if any, is used.

Example of a Multilingual Window

Assume that you have a window that looks like Figure 3:

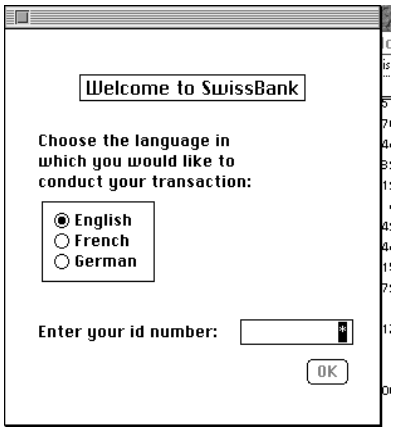


Figure 3 *Sample Window in English*

The following example first checks to see if the language has to be changed by checking the LanguageName attribute on the LocaleDesc class. Assume the application is developed in English.

Assume you have the following constant defined:

```
constant DEFALT_LOCALE = ‘en_us’;
```

```
method LoginWindow.Init()

super.Init();

-- Set current locale name. Use GetLocale() to extract the language
-- and territory parts only from task.LocaleDesc.LocaleName.
self.CurrentLocale = task.GetLocale();

-- Check to see if locale is a supported one.
if self.CurrentLocale <> 'en_us' and
  self.CurrentLocale <> 'fr_fr' and
  self.CurrentLocale <> 'de_de' then
  errMsg : TextData = new(value = self.CurrentLocale);
  errMsg.Concat(NX(' not supported.  Defaulting to '));
  errMsg.Concat(DEFAULT_LOCALE);
  self.Window.MessageDialog(errMsg);
  self.CurrentLocale = DEFAULT_LOCALE;
end if;

-- Set radiolist to highlight the correct start-up choice
if self.CurrentLocale = 'en_us' then
  self.LanguageChoice = 1;
elseif self.CurrentLocale = 'fr_fr' then
  self.LanguageChoice = 2;
elseif self.CurrentLocale = 'de_de' then
  self.LanguageChoice = 3;
```

```
end if;

if self.CurrentLocale <> DEFAULT_LOCALE then
-- make sure to reset
  task.SetLocale(locale = self.CurrentLocale);

-- A special built-in variable, called "application"
-- is defined, which refers to the current
-- ApplicationDesc object for the code.  This object
-- has an attribute called MsgCatalog, which
-- refers to the message catalog for the current
-- project.  This message catalog will reflect the
-- current locale set on the task.

mcat : MsgCatalog = application.MsgCatalog;

-- Calling ReloadLabelText on the Window
-- causes widgets in the form and menu to be
-- reloaded.
self.Window.ReloadLabelText(mcat);
end if;
```

See InternatBank example **Project**: InternatBank • **Class**: LoginWindow • **Method**: Init

The following Display method handles resetting the locale if the user chooses another language:

```
method LoginWindow.Display()

-- English is the default value in the radio list.

requestedLocale : string = DEFAULT_LOCALE;

self.Open();

event loop
  when task.Shutdown do
    exit;

-- The user has selected English, French or German
-- from a RadioList.
when <LanguageChoice>.AfterValueChange do
  case LanguageChoice is
    when 1 do
      requestedLocale = 'en_us';
    when 2 do
      requestedLocale = 'fr_fr';
    when 3 do
      requestedLocale = 'de_de';
  end case;
```

```
-- Check whether the requested locale is different
-- from the current locale for the task.
-- Call the method to reload label text if it is.
if self.CurrentLocale <> requestedLocale then

-- Need to reset the LocaleDesc.
self.CurrentLocale = requestedLocale;
task.SetLocale(locale = self.CurrentLocale);

-- A special built-in variable, called "application"
-- is defined, which refers to the current
-- ApplicationDesc object for the code.  This object
-- has an attribute called MsgCatalog, which
-- refers to the message catalog for the current
-- project.  This message catalog will reflect the
-- current locale set on the task.

mcat : MsgCatalog = application.MsgCatalog;

-- Calling ReloadLabelText on the Window causes
-- widgets in the form and menu to be reloaded.
self.Window.ReloadLabelText(mcat);

end if;

when <IdNumber>.AfterFirstKeystroke do
<OKButton>.State = FS_UPDATE;

when <OKButton>.Click do
accountWindow : AccountWindow = new;
start task accountWindow.Display(
requestedLocale = self.CurrentLocale);

end event;
self.Close();
```

See InternatBank example **Project**: InternatBank • **Class**: LoginWindow • **Method**: Display

Your translated window would look like Figure 4:

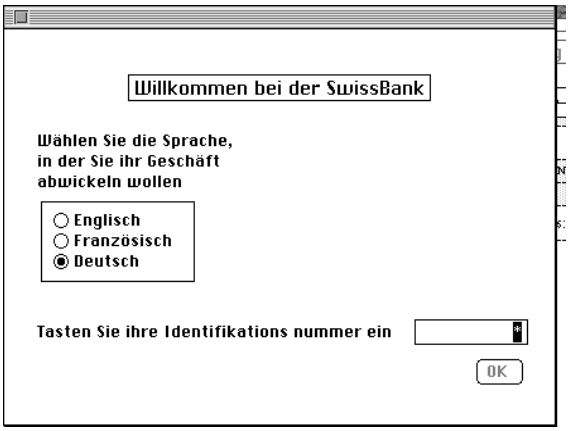


Figure 4 *Sample Window in German*

The following Display method handles the language requirements of the Account Window (second window of the example application):

```
method AccountWindow.Display(input requestedLocale : string)

-- Call the ReloadLabelText method if the language is different
-- from the language the application was developed in.

mcat : MsgCatalog = application.MsgCatalog;

if requestedLocale <> DEFAULT_LOCALE then
self.Window.ReloadLabelText(mcat);
end if;

-- Define a constant for the set number in the
-- catalog file.

constant WINDOWSET = 2;

-- This string variable can be reused by several
-- MessageDialog boxes, as long as GetString is
-- called each time.

myMsg : string;

self.Open();
event loop
when task.shutdown do
exit;
```

```
when <DepositButton>.Click do
myMsg = mcat.GetString
  (setNumber = WINDOWSET,  msgNumber = 8);
self.Window.MessageDialog
  (messageText = myMsg);

-- The text for this message has two parameters.
when <WithdrawButton>.Click do
myMsg = mcat.GetString
  (setNumber = WINDOWSET,  msgNumber = 9,
  defaultString = NX('Branch %1 out of money today: %2.'));

myText : TextData = new;
myText.ReplaceParameters
  (source = myMsg,
  parameter1 = IntegerData(value = 33),
  parameter2 = <DateField>.TextValue);
self.Window.MessageDialog
  (messageText = myText);

end event;
```

See InternatBank example

Project: InternatBank • **Class**: AccountWindow • **Method**: Display

Using Extended Fonts

Forte provides a set of fonts for multinational use. These fonts will vary depending on locale settings. For example, the fonts available to a user of Japanese locale settings will be different than the fonts available for a French locale. As a result, these fonts are not necessarily portable beyond a single codeset or window system type. In fact, most codesets have a set of fonts that have been specifically designed for that codeset. In Japan, for example, a special set of fonts is available (most common is called Mincho) for the standard Kanji set of characters. In Korea, a font representing the Hangul characters is available. These locale-specific fonts are referred to as Forte's extended fonts.

ExtendedFontDescriptor class

The extended fonts are a set of six additional typefaces supported through the ExtendedFontDescriptor class (see the Display Library online Help for more information). Typeface identifiers 1 through 6 designate each typeface, and, as stated above, these typefaces can be different for each of the codesets (such as ISO88591 or SHIFTJIS), and for each supported window system (Windows and Motif).

These new fonts are also available as a set of options within the Font menu in the Window Workshop. The menu items use the descriptive name for the font on the system upon which the user is running. Whenever possible—to assure maximum portability—use the system or portable fonts. These fonts will always work on all systems.

Partitioned International Applications

When you start a client partition, the client uses locale settings to determine codeset translation, language usage, and default formatting. When an application invokes a method on a remote object (in a partition on another machine), that partition may have been started using a different locale (different codeset, different language, and so on). However, because the method invoked on the remote partition is logically part of the client application, some of the locale information is critical to the "natural" operation of the system.

Codeset conversion

Codesets can affect the semantics of an application at runtime. For example, if the string 'Forte' is taken from an NT field and sent to a remote UNIX server to be inserted into a database, the system must confirm that the characters inserted into the database are the same characters typed in by the user, even though the codesets of the two machines may differ. The strings that pass between the partitions must be converted between any mismatching codesets (this is particularly important for the character é). In addition, any error messages returned from the remote partition must be translated to the language being used in the client machine. Because the remote partition may be servicing requests from several clients, which might be running under different locale settings, the remote partition must adapt to the locale information that is defined for the client.

Fortunately, the majority of these conversions take place automatically. For example:

- Conversion between the codeset running in the client machine and the codeset running in the remote machine is handled automatically and transparently.
- The language and territory code for the client partition is passed to the remote partition, so that messages and errors looked up on the remote partition will use that language and territory code.
- The collating sequence from the client partition is passed across such that comparisons and sorts done in the remote partition will use the collating sequence defined for the client.

Passing formatting information across partitions

Forte does not pass the formatting information specified in control panels or environment variables between partitions. Instead, any modification to the locale will apply only to the formatting specifications in the client partition. That is, the formatting will conform as if the server were started with the FORTE_LOCALE set to the language and territory code of the client, using the codeset of the server. To assure that the formatting conventions are exactly as the client has set, you must make sure to do all formatting in the client partition of the application.

Accessing External Resources

As mentioned above, the majority of codeset conversion between partitions takes place automatically. However, there is one case where you will have to alter the startup command of a partition.

External resources, such as databases, often use specific codesets for data storage. If your application accesses a data source that uses a different codeset than the codeset of the partition accessing the data source, you must do one of two things:

- partition your application such that the object that accesses the data source resides on the same partition as the data source

  or

- start up the partition on which the object resides using the same codeset as the partition in which the data source resides

To start up a partition using a particular codeset, include the -flc flag in the Server Arguments field in the Assigned Partition Properties dialog of the Partition Workshop. See *A Guide to the Forte 4GL Workshops* for more information about partition properties.

Typically, a Forte server running against an external resource on a particular machine or network will interact correctly if the resource is codeset aware.

Chapter 3

Specifying Locale Settings

The term *locale* represents properties relating to language and territory conventions. In Windows environments, control panels are available through which users can set locale information. Date and time formats, for example, are typical control panel settings. On UNIX and VMS platforms, users can set default locale information using a single environment variable.

This chapter contains information about Forte locale settings. Topics covered include:

- setting default locales
- customizing locales
- Forte locale environment variables
- precedence rules for locale behavior

Overview

As described in previous chapters, an international application adapts to an environment's locale information—a set of characteristics pertaining to language and territory conventions.

The following table lists the information controlled by a locale:

Category	Description
Language	The language for messages that are seen by the user.
Territory	The territory (usually a country) for other conventions, which determines a set of defaults for currency, money, and date formatting.
Character Handling	Within a codeset, the conventions for classifying characters as upper or lower case, punctuation marks, white space, and so on.
Collating Sequence	Within a codeset, the collating sequence for characters. Most codesets have at least two sequences, one for sorting upper and lower case characters intermixed, and one for sorting uppercase before lowercase.
Currency Format	The default format to use when a numeric format of "CURRENCY" is used. This format is based on a number of settings in the locale definitions and control panels.
Numeric Format	The default formats to use for thousands separators and decimal point. These formats are based on a number of settings in the locale definitions and control panels.
Date and Time Format	The default formats to use when a DateTime format of "DATE", "TIME", or "DATETIME" is used. This is based on a number of settings in the locale definitions and control panels.

Users define locales on a setting-by-setting basis (on platforms with control panels), or by setting an environment variable to activate a single locale definition file. See the section "Setting the Default Locale" on page 51 for information about defining locales.

In addition, end users can modify particular information for their locale using locale environment variables. See the section "Customizing Locales" on page 53 for information about modifying locale information using Forte locale environment variables.

Setting the Default Locale

Most platforms offer their own conventions for setting locale information. Forte's international support takes advantage of industry standard mechanisms whenever possible. In most cases, that mechanism is a control panel or the LANG environment variable. On systems that do not support the LANG variable, Forte provides its own variable, FORTE_LOCALE, which, like LANG, sets up a default locale. Refer to the section "Customizing Locales" on page 53 for more information about the FORTE_LOCALE environment variable.

To set the default locale:

1 Select the appropriate settings in your platform's control panel.

or

1 Set your platform's locale environment variable (LANG or FORTE_LOCALE) to the locale you desire. Check the table below for the variable available on your platform.

c locale

If you do not specify a locale on a UNIX platform, the operating system will use the default locale setting 'c'. A 'c' locale represents the default locale for the operating system. Many operating systems set the LANG environment variable to 'c' as a default. A 'c' locale,' however, tells Forte NOT to perform any codeset conversion. If you are certain you will only need characters found in an ASCII codeset, then the 'c' locale is sufficient. If you use characters outside the ASCII codeset, such as characters with a grave accent or umlaut, then you must specify the appropriate locale. Or, if you have non-ASCII characters in a window, the FORTE_LOCALE environment variable must be set before loading the .pex file. This is not an issue in Windows environments, because a locale is always specified in a control panel; therefore, codeset conversion will always be performed when necessary.

Note

Specific locale support varies between environments, because locale support is an optional feature in most operating systems. Therefore, the examples below may use a locale that your system does not support. Check with your system administrator for the list of available locales on your system.

Platform	Setting	Example
DEC UNIX	LANG	setenv LANG en_US.88591
DGUX	LANG	setenv LANG en_US
HP	LANG	setenv LANG american.iso88591
NT	Control Panel or FORTE_LOCALE	see Figure 5 for Control Panel settings, or, setenv FORTE_LOCALE (in the forte.ini file)
RS/6000	LANG	setenv LANG en_US
Sequent	LANG	setenv LANG en_US
Solaris	LANG	setenv LANG en_US
Sun	FORTE_LOCALE	setenv FORTE_LOCALE en_US.iso
Windows	Control Panel or FORTE_LOCALE	see Figure 5 for Control Panel settings, or, setenv FORTE_LOCALE (in the forte.ini file)
VMS	FORTE_LOCALE	setenv FORTE_LOCALE en_US.iso

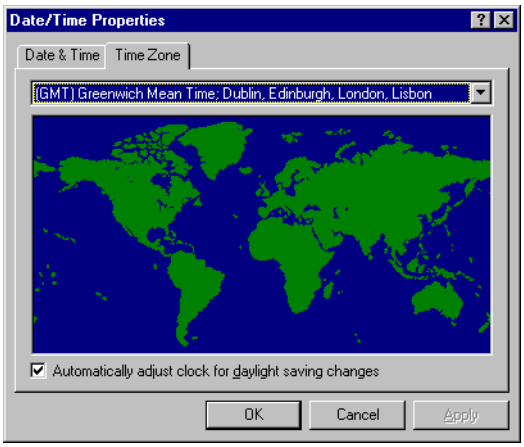


Figure 5 *WindowsNT Date/Time Control Panel*

If you wish to change a particular setting of your default locale, refer to the section"Customizing Locales" on page 53.

Customizing Locales

To offer users flexibility when establishing locales, Forte provides a set of locale environment variables for enhancing or modifying the default locale settings on your system. Most of the time, though, the default locale settings will be sufficient. The table below lists the Forte locale environment variables.

Note X/Open variables are available on most UNIX systems and are synonymous to the Forte variables.

Forte Variable	X/Open Variable	Definition
FORTE_LOCALE	LANG	Sets all of the locale conventions to the information in the Forte specified locale file.
FORTE_LC_CTYPE	LC_CTYPE	Uses the locale you specify for character classifications.
FORTE_LC_COLLATE	LC_COLLATE	Uses the locale you specify for collating sequence information.
FORTE_LC_MONETARY	LC_MONETARY	Uses the locale you specify for currency formatting information.
FORTE_LC_NUMERIC	LC_NUMERIC	Uses the locale you specify for numeric formatting information.
FORTE_LC_TIME	LC_TIME	Uses the locale you specify for date and time formatting information.

The syntax for the Forte locale environment variables is:

variable_name language_territory.codeset[*@collate_sequence*]

The environment variable arguments are described below:

Argument	Description
language_territory	Specifies the language and the territory conventions to use for locale information. See the section "Language and Territory Options" on page 54 for more information.
codeset	Specifies the codeset to use when running any partition. See the section "Codesets and Codeset Conversion" on page 55 for more information.
collate_sequence	Specifies the collating sequence to use within the locale definition file. See the section "Specifying the Collating Sequence" on page 56 for more information.

When you set any of the FORTE_LC_* variables, specify the full locale, although only the category of information appropriate to the environment variable is actually taken from that file. Sample settings of FORTE_LOCALE (in UNIX C-shell format) are shown below:

```
setenv FORTE_LOCALE en_US.mac -- uses all settings
setenv FORTE_LC_MONETARY en_GB.iso -- uses only currency setting
setenv FORTE_LC_CTYPE ja_JA.euc -- character classification settings
setenv FORTE_LOCALE fr_FR.iso@2 -- specific collating sequence
```

Using Forte-Supplied Locales

As mentioned earlier, every platform supports its own set of locales that you specify using the LANG environment variable. There is no guarantee that a particular locale on one platform will reflect exactly the same conventions on another platform. For example, the collating sequence for a French locale on an IBM/RS6000 may be different than the collating sequence for a French locale on an HP.

FORTE_LOCALE

Forte provides its own set of locales, all of which perform exactly the same across platforms. To set these locales, you use the FORTE_LOCALE environment variable. The syntax for setting this variable is the same on every UNIX and VMS platform. The following are examples of setting the FORTE_LOCALE environment variable:

```
setenv FORTE_LOCALE en_US.iso
setenv FORTE_LOCALE fr_CA.iso
setenv FORTE_LOCALE de_DE.iso
```

Available Forte locales are listed in the file $FORTE_ROOT/install/nls/locale.

Language and Territory Options

All Forte locale environment variables use a language_territory argument. The language argument is a two-letter code that specifies which language to use. The language code is followed by an underscore character, followed by a two-letter territory code. The territory code specifies territory-specific conventions, such as how date, time, money, and numeric values are formatted. The set of language_territory (locale) codes are pre-defined by Forte and are listed in the file $FORTE_ROOT/install/nls/locale.

The following table lists a subset of supported, commonly-used locales:

Language/Territory	Locale Code
English—United States	en_us
English—Great Britain	en_gb
English—Australia	en_au
English—Canada	en_ca
French—France	fr_fr
French—Canada	fr_ca
German—Germany	de_de
Italian	it_it
Japanese	ja_jp
Spanish	es_es

Codesets and Codeset Conversion

The codeset argument of a locale environment variable specifies the codeset to use when running any partition. Note that specifying any codeset other than the default for the system can reduce performance with external resources on that system that are running in a different codeset.

The values for codeset are pre-defined by Forte and based on the X/Open standard wherever possible. If you use the Forte variables you can, of course, use Forte's abbreviated codeset names; otherwise, use the native settings.

The following table shows the most common codeset values. Refer to the Release Notes for complete listing of supported codesets:

Short Name	Full Name	Description
iso	ISO8859-1	Standard ISO - Western Europe and Americas
win	WinLatin1	Windows - Western Europe and Americas
dec	DecMulti	Digital - Western Europe and Americas
sjs	Shift-JIS	Japanese Shift-JIS
euc	JaEUC	UNIX extended Japanese codeset

ja_jp codesets

The iso, win, and dec codesets are available for all locales except for ja_jp, which uses sjs, euc, and dec.

Codesets and Distributed Environments

In a distributed environment, partitions may be running in different codesets. Wherever possible, Forte performs codeset conversion automatically and transparently. The following table shows the conversion behavior between platforms:

Codesets	ISO8859-1	MacRom	Win Lat1	DecMulti	Shift-JIS	JaEUC
ASCII	✔	✔	✔	✔	✔	✔
ISO8859-1	✔	●	●	●	—	—
Win Lat1	●	●	✔	●	—	—
DecMulti	●	●	●	✔	—	—
Shift-JIS	—	—	—	—	✔	●
JaEUC	—	—	—	—	●	✔

Legend:
✔ No conversion required
● Automatic conversion by Forte
— Conversion not possible

Note

All codesets are compatible with the ASCII codeset, provided that there are no non-ASCII characters. If there are non-ASCII characters, they will be lost in the conversion.

During codeset conversion, characters in the source codeset that do not have equivalent characters in the destination codeset are usually mapped to characters in the destination codeset that are used as placeholders or are unused slots.

Codesets and Exported .pex Files

If your project includes non-ASCII characters and you export the project into a .pex file, you must be careful to use the right codeset when you import the .pex file. There are two areas where you must be aware of this:

- non-ASCII characters included in TOOL source code, for example, in message text

  In this scenario, you must be using exactly the same codeset when exporting and importing the .pex file.

- non-ASCII characters in window text, for example, in widget labels

  In this scenario, you must be running a compatible codeset when exporting and importing the .pex file.

non-ASCII source code

If you have non-ASCII characters in your TOOL code and plan to export your project to a .pex file and then import it, you must be running Forte in the same codeset you were running when you exported the project.

Note that if you use the Extmsg and Compmsg utilities, the message catalog is tagged with a codeset that allows all characters to be mapped to any compatible codeset.

non-ASCII window labels

If you use the Window Workshop and enter non-ASCII characters in widget labels, such as menus and button names, you must also take care when you export and import the .pex file. When you export the project, the window definition in the .pex file is marked with a codeset identifier. When you import that .pex file, any non-ASCII characters from the base codeset are mapped to the running codeset. If you are running the 'C' codeset (ASCII), it is likely that characters will not be correctly mapped and will be dropped. Therefore, it is important that you specify an explicit locale setting before you import the file.

It is particularly important to set the locale before the import if you are using window inheritance to write multilingual windows.

Specifying the Collating Sequence

This is the numeric identifier for the collating sequence to use within the locale definition file. Values start at 0 (for the first collating sequence). The default is 0, and the maximum is 15.

For example:

```
setenv FORTE_LOCALE en_US.iso@1
setenv FORTE_LOCALE fr_CA.iso@4
```

Settings Precedence for Locale Information

In the vast majority of cases, it is not necessary for an end user to set the FORTE_LC_* environment variables, because the default locale settings and control panel information will cover the majority of situations. In some cases, though, users will supplement or modify the default settings using one or more of the Forte locale environment variables.

When you set environment variables in conjunction with default locale settings, Forte must determine which settings take priority. For example, default locales include currency formats, but when a user sets the FORTE_LC_CURRENCY variable, Forte needs to distinguish which setting is the current setting.

When a partition starts up, the exact resolution of the settings will be done according to the following precedence (from highest precedence first):

Setting	System	What Happens
-flc flag	All systems	If you use the -flc command line flag with the startup command for a partition, Forte consults the specified locale file and uses the settings within that file that pertain to all categories. The locale syntax of the flag is the same as for setting the FORTE_LOCALE environment variable. For example, ftexec -flc en_US.iso
		If any FORTE_LC_* environment variables were set prior to using the -flc flag, Forte ignores them and uses the settings in the locale specified with the -flc flag.
FORTE_LC_* variables	All systems	If you set any of the FORTE_LC_* environment variables, Forte consults the associated locale file and uses the settings within that file that pertain to the category.
		For example, if FORTE_LC_TIME is set to "en_US.iso", then the settings from the locale file for en_US for the iso codeset will be used regardless of what else is set.
FORTE_LOCALE variable	All systems	If you set the FORTE_LOCALE environment variable, Forte consults the specified locale file and uses the settings within that file that pertain to all categories that are not already specified with a FORTE_LC_* environment variable.
LC_* variables	X/OPEN	On X/Open compliant systems that use the LC_* environment variables, Forte uses those variables in the same way as the FORTE_LC_* environment variables to specify the behavior for a group of settings taken from the system locale files. Note that if the FORTE_LOCALE variable is set, none of the LC_* environment variables will take effect, as FORTE_LOCALE has higher precedence.
LANG variable	X/OPEN	On X/Open compliant systems that use the LANG environment variables, Forte uses the LANG environment variables in the same way it uses the FORTE_LOCALE environment variable. Forte consults the specified locale file and uses the settings within that file that pertain to all categories that are not already specified with a LC_* environment variable.
Control Panels	NT and Windows	Forte uses the appropriate control panel settings to collect the information needed to determine the settings for any missing group. This will "fill in" the groups which did not get specifically assigned through the use of a FORTE_LC_* environment variable. Note that if the FORTE_LOCALE variable is set, no information will be taken from the control panels.

On all systems, if Forte cannot determine any of the locale information from the various settings above, or if the specified locale is not available on the system, Forte will print a warning and default to the ASCII codeset and English language with U.S. territory conventions.

Chapter 4

Command Line Utilities

This chapter describes the Extmsg and Compmsg utilities.

The Extmsg Utility

The Extmsg utility takes a TOOL project export file as input and replaces any single-quoted strings found in the text within the file with invocations of the GetString method on the default message catalog for the application. This utility is provided as a simple convenience only, and is in no way required.

The Extmsg utility produces two pieces of output: a text file containing the new TOOL code and a newly created message text source file. You must import the TOOL code back into the repository, and, if necessary, append the message text file information to any existing message file. If you are running the Extmsg utility before you have created any messages, of course, the output message file can act as the starting message file.

The syntax of the Extmsg utility is the following:

Portable syntax

extmsg -i *input_TOOL_file* **-o** *output_TOOL_file*
-m *output_msg_file* [**-s** *set_string*] [**-n** *first_msg_number*]
[**-t** *substitution_text_string*] [**-l**]

OpenVMS syntax

VFORTE EXTMSG
/INPUT_TOOL=*input_TOOL_file*
/OUTPUT_TOOL=*output_TOOL_file*
/MSG_TEXT=*output_msg_file*
[**/SET**=*set_string*]
[**/FIRST_MSG**=*first_msg_number*]
[**/SUBSTITUTE**=*substitution_text_string*]
[**/LEAVE_MSG**=TRUE]

The following table describes the command line flags for the **extmsg** command:

Flag	Description
-i *input_TOOL_file* **/INPUT_TOOL**=*input_TOOL_FILE*	Specifies the name of a text file containing TOOL code from which the strings are to be extracted. The file can be a full export file, or just an arbitrary piece of TOOL code.
-o *output_TOOL_file* /**OUTPUT_TOOL**=*output_TOOL_file*	Specifies the name of the text file that will contain the TOOL code, with all strings replaced by invocations of the GetString method.
-**m** *output_msg_file* /**MSG_TEXT**=*output_msg_file*	Specifies the name of the message text source file that **extmsg** generates, containing the strings that were extracted.
-s *set_string* /**SET**=*set_string*	Specifies a single set number to use for all invocations of the GetString method. The default set number is 1. This flag will also place a set command in the output message text file. This parameter may include a literal string, so you can use a string that might refer to a constant name, for example, -s WINDOWSET=5. To complete this example, you would need to edit the pex file and explicitly include the following constant at the appropriate place: WINDOWSET=5;
-n *first_msg_number* /**FIRST_MSG**=*first_msg_number*	Specifies the first message number within the set that will be used for the invocations of the GetString method. The first message extracted will be given this number, and subsequent extractions will increment the value by 1. The default is 1.
-t *substitution_text_string* /**SUBSTITUTE**=*substitution_text_string*	The text to use to substitute for the leading portion of the GetString method invocation. The default value is: application.MsgCatalog.GetString
-l /**LEAVE_MSG**=TRUE	The single-quoted strings are left as the value of the defaultString parameter. Refer to the Forte online Help for information about the GetString method on the MsgCatalog class. The default is not to leave the strings for the parameter.

The following example demonstrates the use of the Extmsg utility. Assume that you had the following TOOL code in a file called myin.tol:

```
method classX.methodY
  m : TextData = new;
  m.SetValue('Preparing long running query...');
  self.Window.MessageDialog(message = m);
  sql select into my_array from my_table
    where my_table.Name = NX('Jones');
  m.SetValue('Done with query.');
  self.Window.MessageDialog(message = m);
end method;
```

You could run the following command:

```
extmsg -i myin.tol -o myout.tol -m myout.msg -s 4 -f 102 -l
   -t application.MsgCatalog.GetString
```

The command above would generate the output file myout.tol:

```
method classX.methodY
  m : TextData = new;
  m.SetValue(application.MsgCatalog.GetString(4, 102, 'Preparing
     long running query...');
  self.Window.MessageDialog(message = m);
  sql select into my_array from my_table
    where my_table.Name = NX('Jones');
  m.SetValue(application.MsgCatalog.GetString(4, 103, 'Done with
    query.');
  self.Window.MessageDialog(message = m);
end method;
```

The output file would look like the following:

```
$set 4
102 Preparing long running query...
103 Done with query.
```

Preventing String Extraction

The Extmsg utility has no knowledge of TOOL syntax, so it will translate all strings it finds in TOOL code, excluding comments. To prevent extraction unnecessary strings, Forte provides a special TOOL function that takes the form NX(*string_literal*). NX is short for No Translation.

Specifying Message Catalog Numbering Sequence

You can use the -**s** and -**f** parameters to tell Extmsg to use a specific set number string and starting message number within the set when it extracts the string messages. By default, it uses set number 1, starting at message number 1.

The Compmsg Utility

The Compmsg utility takes a file containing source message text and numbers, and compiles it into a portable binary format for use at runtime by the Forte runtime system. You must run the Compmsg utility whenever you add or change the messages in the message source text file.

The syntax of the Compmsg utility is the following:

Portable syntax

compmsg -m [**@**]*input_msg_file* **-c** *output_msg_catalog* [**-o**] [**-d** *msg_catalog]*
[-**e** *log_file*] [-**flc** *locale*]

VMS syntax

VFORTE COMPMSG
/**MSG_TEXT**=*input_msg_file*
/**CATALOG**=*output_msg_catalog*
[/**OVERWRITE**=TRUE]
[/**DISPLAY**=*msg_catalog*]
[/**ERROR_LOGFILE**=*log_file*]

The following table describes the command line flags for the **compmsg** command:

Flag	Description
-m [**@**]*input_msg_file* /**MSG_TEXT**=*input_msg_file*	Specifies the name of the source message text file to compile. To support a batch input file, preface *input_msg_file* with an '**@**'.
-c *output_msg_catalog* /**CATALOG**=*output_msg_catalog*	Specifies the name of the portable binary message catalog file to create.
-o	Overwrites the *output_msg_catalog* file if it exists. Otherwise, **compmsg** merges *input_msg_file* with *output_msg_catalog*.
-**d** *msg_catalog* /**DISPLAY**=*msg_catalog*	Displays the contents of the compiled catalog *msg_catalog*, allowing you to reconstruct the message file from the catalog.
-e *log_file* /**ERROR_LOGFILE**=*log_file*	Specifies the name of the log file to use for capturing all output messages/diagnostics from Compmsg. This is particularly useful when used together with the /**DISPLAY**=*msg_catalog* to capture the output to a file.
-**flc** *locale*	Overrides the FORTE_LOCALE or control panel setting.This allows you to maximize performance by running Compmsg with the locale specific to the target environment. For example, if you are using Compmsg on a PC, but compiling a file for use on a Macintosh, use a locale with a .mac suffix. The -**flc** flag has no effect if the -**d** option is also specified.

The message catalog generated by the Compmsg utility is completely portable across operating systems and codesets. For maximum performance, however, you might want to compile separately on machines that use different native codesets to avoid runtime codeset conversion when the message is read.

Alternatively, you can use the -**flc** flag to override the LANG or FORTE_LOCALE setting on the current machine. In this way you can compile the application specifically for its destination codeset. For example, the following table shows example uses of the -**flc** flag for specific codesets (note that the *language_territory* specification is not relative to the codeset, but is included for completeness):

Message catalog contains...	Use this command line option
ASCII characters only	-flc C
ASCII and ISO8859-1 characters	-flc en_US.iso
ASCII and DEC MultiNational characters	-flc en_US.dec
ASCII and Windows Latin 1 characters	-flc en_US.win
ASCII and Japanese Shift-JIS characters	-flc ja_JP.sjs
ASCII and Japanese EUC characters	-flc ja_JP euc

The -**flc** option has no effect when the -**d** option is in use. In other words, if you are displaying the contents of a message catalog, the contents will always be displayed in the codeset in which the catalog was originally created.

Index

G

H

I

L

M

N

O

P

S

T

U

W